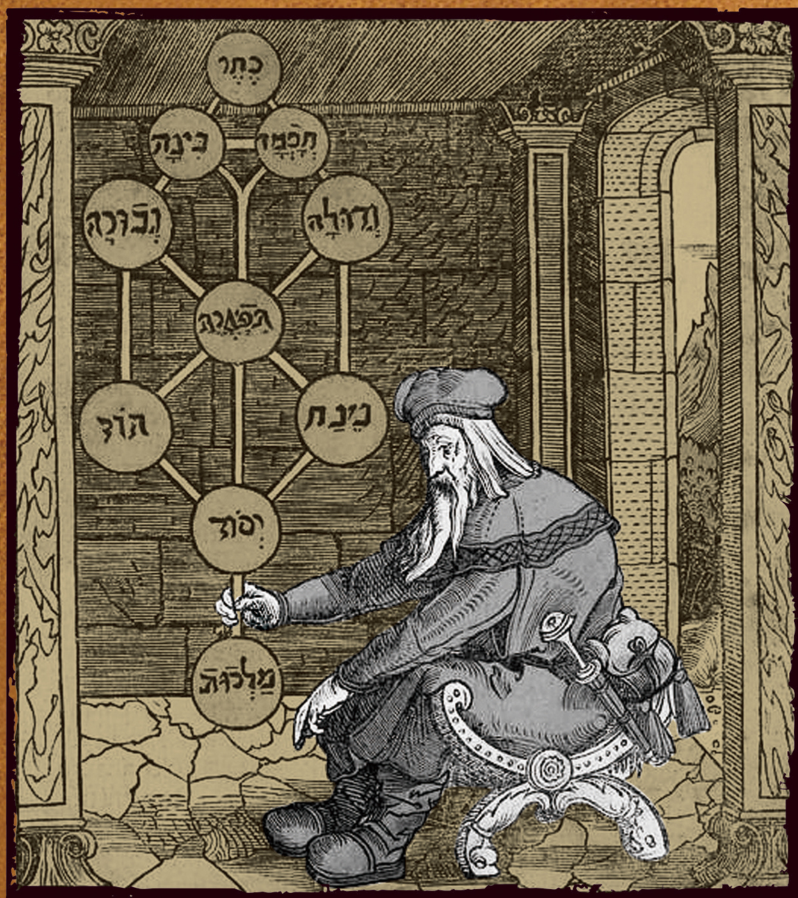
TURING 图灵原创

深入理解神经网络

从逻辑回归到CNN

张觉非 ◎著



 中国工信出版集团  人民邮电出版社
POSTS & TELECOM PRESS



— 张觉非 —

本科毕业于复旦大学计算机系，

于中国科学院古脊椎动物与

古人类研究所取得古生物学硕士学位，

目前在互联网行业

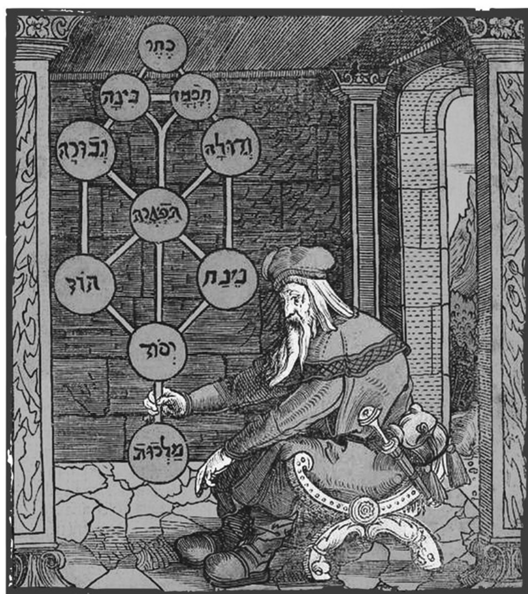
从事机器学习算法相关工作。

数字版权声明

深入理解神经网络

从逻辑回归到CNN

张觉非◎著



人民邮电出版社

北京

图书在版编目（CIP）数据

深入理解神经网络 : 从逻辑回归到CNN / 张觉非著.
-- 北京 : 人民邮电出版社, 2019.9
（图灵原创）
ISBN 978-7-115-51723-4

Ⅰ. ①深… Ⅱ. ①张… Ⅲ. ①人工神经网络一研究
Ⅳ. ①TP183

中国版本图书馆CIP数据核字(2019)第155659号

内 容 提 要

本书以神经网络为线索，沿着从线性模型到深度学习的路线讲解神经网络的原理和实现。本书将数学基础知识与机器学习和神经网络紧密结合，包含线性模型的结构与局限、损失函数、基于一阶和二阶信息的优化算法、模型自由度与正则化、神经网络的表达能力、反向传播与计算图自动求导、卷积神经网络等主题，帮助读者建立基于数学原理的较深刻的洞见和认知。本书还提供了逻辑回归、多层全连接神经网络和多种训练算法的 Python 实现，以及运用 TensorFlow 搭建和训练多种卷积神经网络的代码实例。

本书适合渴望加深对神经网络和深度学习原理理解的高年级本科生与研究生、广大程序员与工程师，以及对机器学习的原理和编程实现感兴趣的所有读者阅读。

◆ 著　　　　张觉非
　责任编辑　陈兴璐
　责任印制　周昇亮
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
　邮编　100164　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　　印刷
◆ 开本：800×1000　1/16
　印张：20.25
　字数：479千字　　　　　2019年9月第1版
　印数：1－3 000册　　　　2019年9月北京第1次印刷

定价：89.00元

读者服务热线：(010)51095183转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

献给周怡萍女士，我生命中的奇异吸引子。

前　　言

“理论是灰色的，我亲爱的朋友，而生命的金树长青。”

——歌德，《浮士德》

近年来，深度学习的浪潮席卷了学术界和产业界，它强大的能力给人们留下了深刻的印象。在产业界，一些已经成熟应用机器学习技术的领域，如计算广告、推荐系统、机器视觉、自然语言处理等，纷纷尝试深度学习并取得了良好的效果。各个尚未应用机器学习的领域也都在积极进行智能化的尝试。另外，开源社区贡献的众多框架、工具和平台，也为这股浪潮推波助澜。可以说，机器学习，尤其是神经网络与深度学习，是当今最被寄予厚望的技术。

不论这股浪潮将止于何处，也不论它最终能否达到人们的期许，机器学习的思想都已经深深植入了人们的心中。在工程和业务实践中，将数据和建模纳入解决方案已是一种常规做法。机器学习已成为工程师武器库中一个重要的工具，越来越多工程师开始尝试了解和应用机器学习技术。

但是人们会发现，机器学习有着艰深的理论背景，不能透彻理解原理，就难以在实践中自由地运用。当人们试图进入机器学习领域时，数学原理是横亘在面前的一座大山。笔者通过自身经历以及与他人的交流，深切体会到，广大学生和工程师在基础理论知识的理解和运用上普遍有所欠缺，这给他们理解机器学习的原理制造了障碍。

举个例子，对于梯度和链式法则，许多人只能在一元函数情况下理解，大量文章和书籍也都只在一元情况下做浅尝辄止的讲解。而当人们阅读代码时，会发现其中满是矩阵和向量。原理与实现之间仿佛弥漫着一团迷雾，不穿透这团迷雾，对这些概念的理解就只能是浮光掠影、隔靴搔痒。笔者曾读到有文章将梯度下降的矩阵实现称为“向量化”。其实何须向量化，多元函数的梯度与求导原本就是用矩阵和向量语言来描述的。

再比如，神经网络与深度学习的火热虽然给机器学习这门学科带来进步和活力，但是也给人们造成一个错误的印象，即它是全新的、颠覆性的、与传统割裂的。其实神经网络与深度学习深

深扎根于传统机器学习之中。读者可以看到，本书的大量篇幅都在讲解线性模型，这是因为神经网络与深度学习的结构和训练方法都扎根于线性模型之中。

目标读者

笔者遇到过不少理工科相关专业的工程师，他们学习过微积分、线性代数和概率论等基础课程，但是对这些知识的理解尚没有融会贯通，不知如何运用，而且搁置多年，遗忘严重。还有一些高年级本科生和研究生感觉各门基础课程仿佛是分散的、孤立的，不理解它们的联系和用途。本书的目标就是帮助这些读者回忆并夯实基础知识，深入理解机器学习，特别是神经网络和深度学习的原理。

本书以神经网络为线索，沿着从线性模型到深度学习的路线，串起核心知识点。数学内容在必要和恰当的时机引入，从基础讲起，不留黑盒。具有理工科背景的读者能够容易地回忆起相关知识，而不需要再回头求诸于大部头教科书。本书内容取舍有当，紧紧围绕主题，使读者能将数学与它在机器学习中的应用紧密结合起来。

本书试图帮助读者打通关节，提高视角，加深洞见，做到知其然并知其所以然。在当今人工智能时代，新方法不断涌现，新工具层出不穷，但是万变不离其宗，新的方法和工具都深深扎根于原理之中。把众多方法比作岛屿，在水面之下看，大大小小的岛屿就不再是孤立的、个别的，它们都建筑于基岩之上，基岩就是机器学习的基础理论。掌握了基础理论，就可以对各种方法形成统一而深刻的洞见，在工程实践中拥有坚实有效的理论依据，并能够快速理解和运用业界新涌现的众多方法和工具。

具体来说，本书适合以下几类读者：

- 渴望进入这一领域的高年级本科生和研究生，本书帮助这类读者将数学基础知识与机器学习和神经网络结合起来，深入理解原理；
- 希望了解神经网络与机器学习的广大程序员与工程师，这类读者需要花更多一些力气回忆数学知识，本书包含了他们需要的全部知识点；
- 对模型的编程实现感兴趣的读者，本书包含逻辑回归、多层全连接神经网络以及各种训练算法的 Python 实现，可供这类读者学习和参考；
- 业界的机器学习、数据挖掘工程师，本书关于模型原理的高级主题，特别是关于模型自由度与偏置-方差权衡方面的内容，可为他们提供一些有趣的洞见。

明确本书不包含的内容以及不适合的读者同样重要，以下是本书不涉及的内容：

- 本书围绕着神经网络这个主题，并涵盖了机器学习学科的相当一部分重点内容，但并不涵盖机器学习的全部领域；
- 本书包括逻辑回归、多层全连接神经网络以及多种训练算法的 Python 实现，但书中代码实现的目的是为了理解原理，本书并不是一本实现各类机器学习模型的指南；
- 本书包含使用 TensorFlow 搭建并训练模型的实例，但本书远远不是一本 TensorFlow 的实用教材；
- 本书从原理上讲解了模型超参数、自由度、过拟合与欠拟合等概念，但本书并非关于调参和优化模型的实践指南，也不包含特征预处理、特征工程等方面的内容；
- 本书以卷积神经网络为例讲解深度学习，涵盖了训练深度神经网络的方法、问题以及技术，但本书并不包含深度学习众多五花八门的新应用领域和网络结构。

内容概览

本书沿着从线性模型到神经网络，再到深度学习的路径，层层递进，逐渐深入。这是一条从根至叶、自简入繁的天然路径。各章节的安排如下。

- 第 1 章以逻辑回归为例讲解线性模型。这一章在引入逻辑回归的定义后回顾必要的向量几何，使读者对逻辑回归的原理、特性和局限形成清晰的认识。
- 第 2 章介绍模型训练和评价的基本概念，并引入损失函数。损失函数将模型训练问题转化为函数优化问题。这一章以交叉熵为例讲解损失函数，并从信息论、贝叶斯和几何特性三个角度探究交叉熵的原理。
- 第 3 章介绍基于函数局部一阶信息的优化算法。这一章首先回顾多元函数微分的相关知识并引入梯度概念，接着讲解梯度下降法及其变体，最后介绍如何用梯度下降法训练逻辑回归模型。
- 第 4 章介绍基于函数局部二阶信息的优化算法。这一章首先回顾矩阵的相关知识，接着讲解多元函数的赫森矩阵，以牛顿法和共轭方向法为例介绍二阶优化算法，最后介绍如何用牛顿法训练逻辑回归模型。
- 第 5 章首先回顾概率论相关知识，之后从线性回归入手，介绍模型自由度、正则化和偏置–方差权衡等概念，最后介绍如何在逻辑回归的训练中应用正则化。
- 第 6 章讲解如何将线性模型连接成网络以克服其局限，并介绍多层全连接神经网络。
- 第 7 章讲解训练多层全连接神经网络的反向传播加梯度下降法。这一章是对之前各章节知识的一个综合，最后还谈到了训练深层网络所面临的一些困难。

- 第 8 章介绍计算图和自动求导。深度学习中的各种网络具有比多层全连接神经网络复杂得多的连接方式，而计算图和自动求导是搭建和训练复杂网络结构的有力工具。
- 第 9 章介绍卷积神经网络的原理、结构和训练。深度神经网络五花八门，卷积神经网络是其中最典型、最具代表性的一种。理解了卷积神经网络及其训练，也就能够理解其他各种深度神经网络。
- 第 10 章介绍了 5 种经典的卷积神经网络，介绍它们的结构和性能，最后简单讨论卷积神经网络结构的发展演化趋势。
- 第 11 章展示如何使用 TensorFlow 搭建和训练本书提到的几种主要模型，并将它们用于手写数字识别。
- 在附录中，我们尝试在卷积神经网络和元胞自动机之间建立联系，从动力学角度理解“深度”的含义。元胞自动机是一类计算模型，它们中的一些具备图灵完备性。元胞自动机的计算方式与卷积具有相似性，这一章从元胞自动机的动力学特性的视角看卷积神经网络，希望为读者提供一些有趣的洞见。

阅读方式

本书各章之间存在依赖关系，但对于不同背景和需求的读者，可以选择不同的阅读路径。

- 数学基础过硬的读者可以略过 1.2 节、3.1 节、4.1 节、5.1 节和 7.1 节，但温习一下总是有益的。
- 熟悉传统机器学习，想进一步了解神经网络和深度学习的读者，可以略过第一部分。
- 对于第 4 章，读者可以略过 4.1 节之后的内容，因为后续章节没有用到二阶优化算法。但 4.1 节回顾的矩阵知识在后续章节中还将用到，且非常关键。

本书各章节组成一个有机的整体，而并非孤立存在。本书路线图中的知识点前后呼应，能为读者提供一些有趣而深刻的洞见。所以，笔者强烈建议读者按顺序阅读全部内容。

代码与网络资源

读者可在 gitee.com/neural_network/neural_network_code 找到本书的样例代码，后续我们将继续维护和扩展这个代码库。笔者在知乎开设了专栏“计算主义”（zhuanlan.zhihu.com/pillgrim），以后将继续在专栏中讨论和分享本书相关主题。鉴于笔者水平有限，错误难免，欢迎读者来信指正，邮箱是 zhangjuefei83@163.com。

致谢

感谢 360 智能工程部“Prophet 机器学习平台”团队的同事：组长陈震、黄斌、林灯博士、孙晓冬、马宗超、吕仁杰、刘韦菠、蔡春蒙、缪路文。孙晓冬为本书第 11 章编写了样例代码，感谢他杰出的工作。

感谢北京图灵文化发展有限公司的陈兴璐女士，她的邀约和鼓励促使本书得以面世，她出色的编辑工作使本书避免了很多问题和缺陷。

感谢中国科学院古脊椎动物与古人类研究所的潘雷博士和北京理工大学生物医学工程系的陈端端教授，她们在参考文献方面给予极大的帮助。

感谢中国科学院古脊椎动物与古人类研究所副研究员朱幼安博士和上海社会科学院哲学研究所钱立卿博士，感谢他们多年来与我在生命演化、混沌系统和计算理论方面的共同兴趣和探讨。

感谢北京自然博物馆标本部副研究员刘迪，蒙他惠允于本书中使用鸟类骨骼与生态类群数据集。

最后，感谢我的家人。

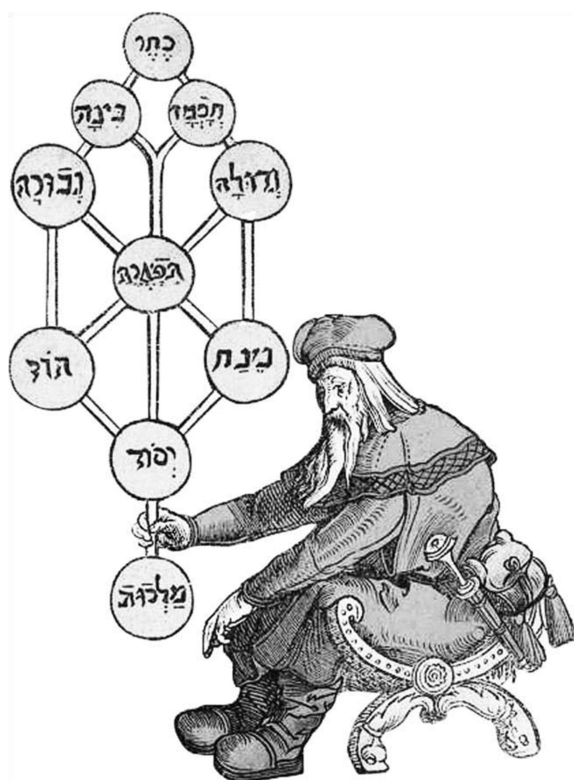
目　录

第一部分　线性模型

第二部分 神经网络

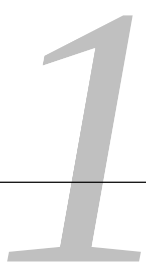
第一部分

线性模型



第 1 章

逻辑回归



统计学习领域的泰斗 Trevor Hastie 曾经说过："每一个机器学习研究者都应该像熟悉自家后院那样熟悉线性模型。"线性模型是许多非线性模型的基础，它良好的数学结构能够为理解非线性模型提供深刻的洞见。人工神经网络的基本组成单元——神经元，正是线性模型。本章将介绍一种常用的线性模型——逻辑回归：首先引入逻辑回归的定义，之后回顾必要的线性代数知识，并从几何角度阐述逻辑回归及所有线性模型的能力局限。

1.1　作为一个神经元的逻辑回归

我们以逻辑回归（logistic regression）为例讨论线性模型。假如样本由n个数值型特征$x_1, x_2, \cdots, x_n$组成，则逻辑回归的计算式是：

$$f(x_1, x_2, \cdots, x_n) = \frac{1}{1+\mathrm{e}^{-(b+w_1x_1+w_2x_2+\cdots+w_nx_n)}} = \frac{1}{1+\mathrm{e}^{-\left(b+\sum_{i=1}^{n} w_ix_i\right)}} \tag{1.1}$$

其中，e是自然对数的底。式（1.1）将各个特征x_i乘以对应的权重系数w_i后相加，之后再加上b，b称作偏置（bias）。这种计算称为关于$x_1, x_2, \cdots, x_n$的仿射函数（affine function），仿射函数的值a如下：

$$a = b + \sum_{i=1}^{n} w_ix_i \tag{1.2}$$

接下来，逻辑回归对仿射函数的值a施加 Logistic 函数：

$$\mathrm{Logistic}(a) = \frac{1}{1+\mathrm{e}^{-a}} \tag{1.3}$$

Logistic 函数的图像是一条 S 形曲线（也称 sigmoid 曲线）。当a趋向于负无穷时，函数值趋向于 0；当a趋向于正无穷时，函数值趋向于 1。Logistic 的函数值在$(0,1)$内，当$a = 0$时，函数值是 0.5，如图 1-1 所示。

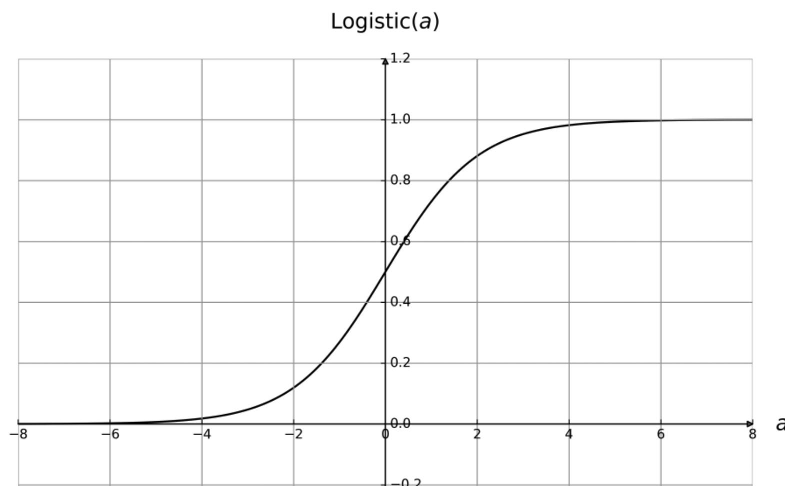


图 1-1 Logistic 函数的图像

在二分类问题中，用n个数值型特征表示一个样本，例如用体重、身高和年龄表示一个人。这些样本属于两个互斥的类别——正类（positive）或负类（negative）。逻辑回归的输出值可视作样本属于正类的概率p_{p}：

$$p_{\mathrm{p}}=\frac{1}{1+\mathrm{e}^{-a}}=\frac{1}{1+\mathrm{e}^{-\left(b+\sum_{i=1}^{n}w_i x_i\right)}} \tag{1.4}$$

因为样本必属于正类或负类之一，而且不能既属于正类又属于负类，所以样本属于正类的概率与属于负类的概率之和为 1，于是样本属于负类的概率p_{n}就是：

$$p_{\mathrm{n}}=1-p_{\mathrm{p}}=1-\frac{1}{1+\mathrm{e}^{-a}}=\frac{\mathrm{e}^{-a}}{1+\mathrm{e}^{-a}} \tag{1.5}$$

p_{p}与p_{n}之比的对数（除非特殊说明，本书中对数都以e为底）是：

$$\log\frac{p_{\mathrm{p}}}{p_{\mathrm{n}}}=\log\frac{1}{1+\mathrm{e}^{-a}}\cdot\frac{1+\mathrm{e}^{-a}}{\mathrm{e}^{-a}}=\log\mathrm{e}^{a}=a=b+\sum_{i=1}^{n}w_i x_i \tag{1.6}$$

可见，逻辑回归模型中两类别的对数概率比是关于特征的仿射函数。我们可以根据概率判定样本类别，如果规定当$p_{\mathrm{p}}\geqslant\frac{1}{2}$，即$\frac{p_{\mathrm{p}}}{p_{\mathrm{n}}}\geqslant 1$时，将样本判定为正类，那么根据式（1.6）有：

$$a=\log\frac{p_{\mathrm{p}}}{p_{\mathrm{n}}}\geqslant\log 1=0 \tag{1.7}$$

也就是说，以$\frac{1}{2}$为概率阈值时，预测类别取决于a：a大于等于 0 时，预测为正类，否则预测为负类。a大于 0 有什么几何意义？逻辑回归为什么属于线性模型？答案将在下文揭晓。在那之前，我们先看看逻辑回归的一种图示，如图 1-2 所示。

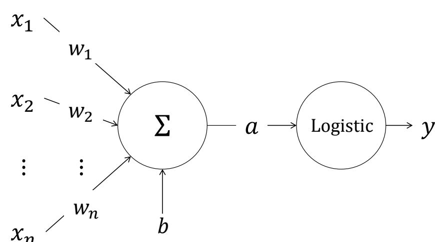


图 1-2　逻辑回归的神经元表示

图 1-2 表达的就是逻辑回归：将输入值加权求和再加偏置之后施加 Logistic 函数。在人工神经网络（本书后面会省略“人工”二字）语境下，这个结构就是一个神经元。这里的 Logistic 函数称为激活函数（activation function）。除了 Logistic 函数外，还有许多其他种类的激活函数，例如阶跃函数：

$$f(x)=\begin{cases}0,\ x<0\\1,\ x\geqslant 0\end{cases} \tag{1.8}$$

以阶跃函数为激活函数的神经元是最早的神经元模型——感知机（perceptron）。感知机这个名称一直流传下来，以至于今天多层全连接神经网络还被称为多层感知机（multilayer perceptron，MLP）。关于激活函数的类型和性质，第 6 章还有阐述。无论取哪种激活函数，神经元的基本结构都是仿射函数加激活函数。

1.2　基础向量几何

神经元是神经网络乃至深度学习的基石，在深入考察神经元的能力和局限之前，有必要回顾一下基础的向量几何。本节概述向量知识。像这样介绍数学基础的小节会在书中多次出现，它们穿插在章节之中，紧密与主题结合，可以帮助读者回忆相关知识点，加深理解。虽是概览，但本书对数学内容的介绍会尽量做到穷根究底，不留黑盒，每一个结论都给出说明。

1.2.1　向量

一个包含n个数值型特征$x_1,x_2,\cdots,x_n$的样本可用n维向量表示：

$$\boldsymbol{x}=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}=(x_1,x_2,\cdots,x_n)^{\mathrm{T}} \tag{1.9}$$

本书用黑斜体小写字母表示向量，例如$\boldsymbol{x}$和$\boldsymbol{y}$。斜体小写字母表示标量（实数），例如a、b或

x。用脚标表示向量的分量，例如x_2表示向量$\boldsymbol{x}$的第 2 个分量。用上标表示一组对象中的某一个，例如$\boldsymbol{x}^3$表示一组向量中的第 3 个。样本有多少特征，$\boldsymbol{x}$就有多少分量。n是向量的分量个数，称为向量的维数。本书中向量专指列向量（其分量竖着排列）。有时为了节省空间，把列向量写成行向量的转置，如式（1.9）。转置就是将行向量变成列向量，或将列向量变成行向量。

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}^{\mathrm{T}} = (x_1, x_2, \cdots, x_n)，\quad (x_1, x_2, \cdots, x_n)^{\mathrm{T}} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{1.10}$$

为了方便查看，本节以 2 维或 3 维向量为例来介绍。现实问题中向量的维数会远远超过 3，但是本节的理论结果可以扩展到任意维数。

如果将各个分量看作坐标值，那么向量$\boldsymbol{x}$可以表示坐标系上的一个点（point）。向量$\boldsymbol{x}$也可以看作从原点指向这个点的一个有长度和方向的“箭头”。向量包含方向和长度这两个信息，点和箭头都是向量的几何表现形式，如图 1-3 所示。论述中可根据讨论视角的不同采用不同的表现形式。

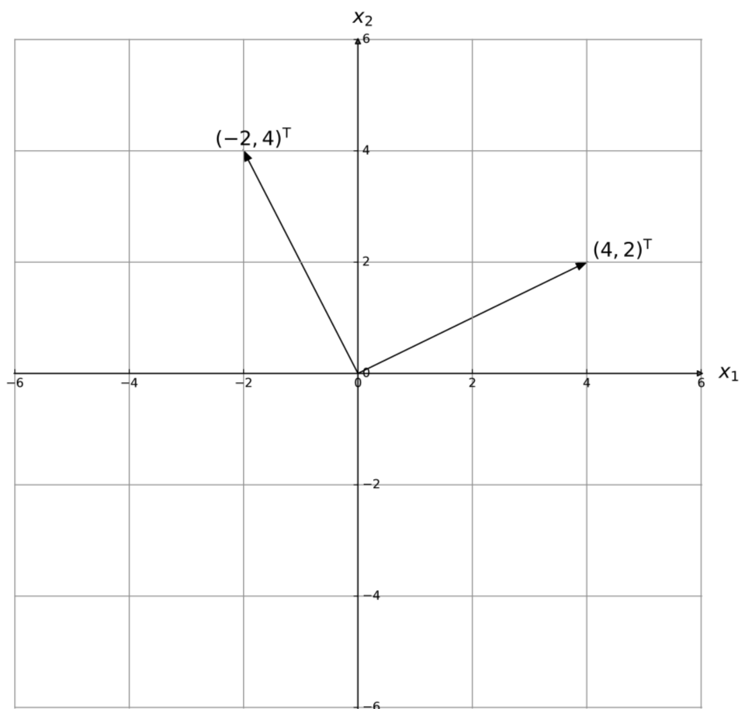


图 1-3　向量的几何表示

1.2.2　向量的和、数乘与零向量

向量可以求和。令$\boldsymbol{x}$和$\boldsymbol{y}$是两个维数相同的向量，它们的和$\boldsymbol{x}+\boldsymbol{y}$是一个向量：

$$\boldsymbol{x}+\boldsymbol{y}=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}+\begin{pmatrix}y_1\\y_2\\\vdots\\y_n\end{pmatrix}=\begin{pmatrix}x_1+y_1\\x_2+y_2\\\vdots\\x_n+y_n\end{pmatrix} \tag{1.11}$$

$\boldsymbol{x}+\boldsymbol{y}$的各分量是$\boldsymbol{x}$和$\boldsymbol{y}$的对应分量之和。如果用“箭头”表示向量，那么$\boldsymbol{x}+\boldsymbol{y}$是以$\boldsymbol{x}$和$\boldsymbol{y}$为邻边组成的平行四边形的对角线，从原点指向相对的顶点。该顶点也就是向量$\boldsymbol{x}+\boldsymbol{y}$表示的点，如图 1-4 所示。

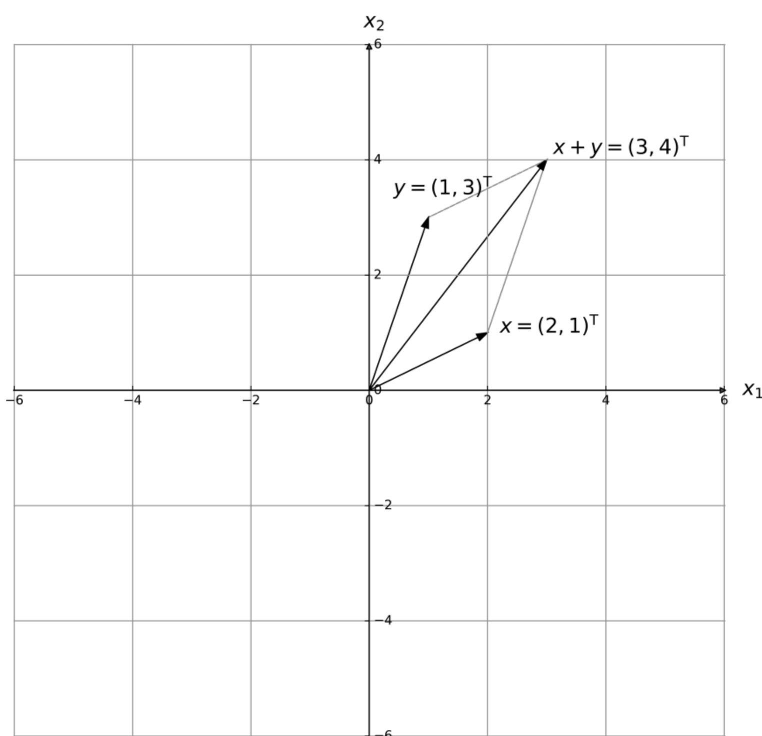


图 1-4　向量和的几何表示

用一个标量（实数）k乘以一个向量，即向量的数乘。定义$k\boldsymbol{x}$为：

$$k\boldsymbol{x}=\begin{pmatrix}kx_1\\kx_2\\\vdots\\kx_n\end{pmatrix} \tag{1.12}$$

向量数乘的几何意义是对向量的长度进行缩放。用 2 乘以向量$\boldsymbol{x}$得到 $2\boldsymbol{x}$，$2\boldsymbol{x}$与$\boldsymbol{x}$方向相同，长度是$\boldsymbol{x}$的 2 倍。向量的数乘如图 1-5 所示。

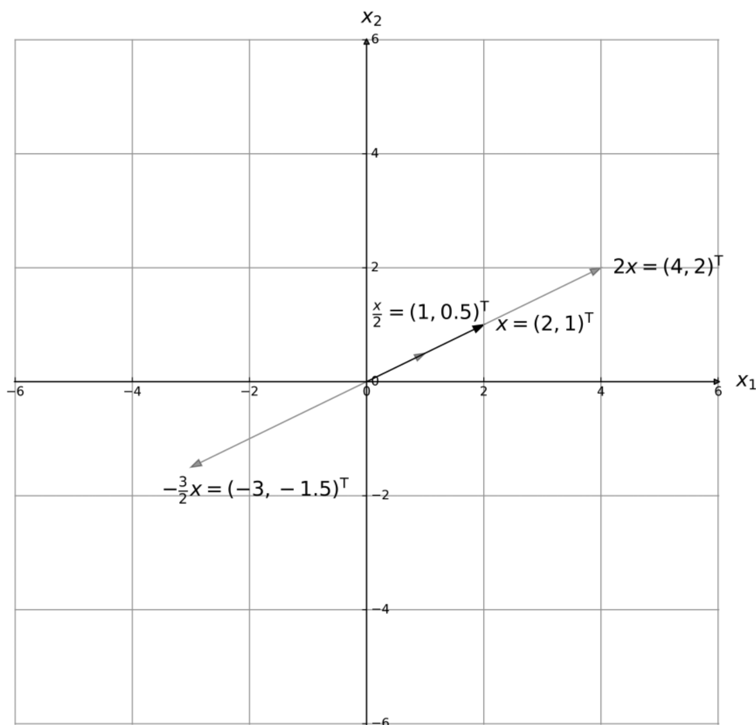


图 1-5 向量的数乘——缩放

用 0 乘以任何向量得到“零向量”：

$$\mathbf{0} = 0 \times \boldsymbol{x} = \begin{pmatrix} 0x_1 \\ 0x_2 \\ \vdots \\ 0x_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{1.13}$$

零向量是坐标系原点。我们很容易看出，零向量与任何向量相加都得到该向量本身。如果用 −1 乘以向量$\boldsymbol{x}$，得到：

$$(-1) \times \boldsymbol{x} = -\boldsymbol{x} = \begin{pmatrix} -x_1 \\ -x_2 \\ \vdots \\ -x_n \end{pmatrix} \tag{1.14}$$

显然，$-\boldsymbol{x}$与$\boldsymbol{x}$相加得到零向量。用$-\boldsymbol{x}$可以定义向量的减法：

$$\boldsymbol{y}-\boldsymbol{x}=\boldsymbol{y}+(-\boldsymbol{x}) \tag{1.15}$$

将箭头$\boldsymbol{y}-\boldsymbol{x}$平移，使其尾部与$\boldsymbol{x}$重合，头部与$\boldsymbol{y}$重合，则$\boldsymbol{y}-\boldsymbol{x}$构成以$\boldsymbol{x}$和$\boldsymbol{y}$为邻边的三角形的第三边，如图 1-6 所示。

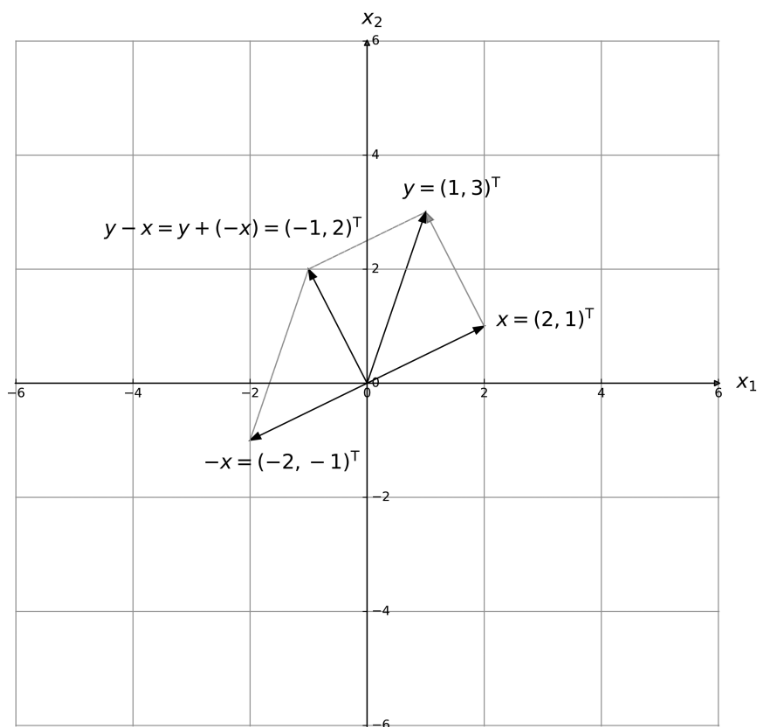


图 1-6　向量的差——三角形的第三边

向量有长度，2 维向量的长度是：

$$\mathrm{length}(\boldsymbol{x})=\sqrt{{x_1}^2+{x_2}^2} \tag{1.16}$$

这个长度是向量与原点之间的欧式距离。3 维乃至更高维向量的长度也是它们与原点之间的欧氏距离——各分量平方和的平方根。

1.2.3　向量的内积、模与投影

向量$\boldsymbol{x}$和$\boldsymbol{y}$的内积（inner product）定义为：

$$\langle\boldsymbol{x},\boldsymbol{y}\rangle=\sum_{i=1}^{n}x_i y_i \tag{1.17}$$

$\boldsymbol{x}$和$\boldsymbol{y}$的内积就是将$\boldsymbol{x}$和$\boldsymbol{y}$的对应分量相乘再相加。根据式（1.17），$\boldsymbol{x}$与自身的内积是：

$$\langle \boldsymbol{x}, \boldsymbol{x} \rangle = \sum_{i=1}^{n} x_i^{\,2} \tag{1.18}$$

所以$\boldsymbol{x}$与自身的内积一定大于等于 0。只有当$\boldsymbol{x}$的所有分量都是 0，即$\boldsymbol{x}$为零向量时，$\boldsymbol{x}$与自身的内积才为 0。向量内积满足交换律：

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle \tag{1.19}$$

向量内积对向量加法满足分配律：

$$\langle \boldsymbol{w}, \boldsymbol{x} + \boldsymbol{y} \rangle = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \langle \boldsymbol{w}, \boldsymbol{y} \rangle \tag{1.20}$$

向量内积对向量数乘满足：

$$\langle k\boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, k\boldsymbol{x} \rangle = k\langle \boldsymbol{x}, \boldsymbol{y} \rangle \tag{1.21}$$

以上三条定律根据向量内积的定义很容易证明，本书从略。向量$\boldsymbol{x}$的模$\|\boldsymbol{x}\|$定义为$\boldsymbol{x}$与自身的内积的平方根：

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} = \sqrt{\sum_{i=1}^{n} x_i^{\,2}} \tag{1.22}$$

可以看出，向量的模就是它的长度。根据定义，$k\boldsymbol{x}$的模是：

$$\|k\boldsymbol{x}\| = \sqrt{\sum_{i=1}^{n} k^2 x_i^{\,2}} = \sqrt{k^2 \sum_{i=1}^{n} x_i^{\,2}} = |k|\|\boldsymbol{x}\| \tag{1.23}$$

根据式（1.23），用标量$\frac{1}{\|\boldsymbol{x}\|}$乘以$\boldsymbol{x}$，得到的向量$\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$的长度是 1，其方向与$\boldsymbol{x}$一致。也就是说，用标量$\frac{1}{\|\boldsymbol{x}\|}$乘以$\boldsymbol{x}$的效果是保持$\boldsymbol{x}$的方向不变，将它的长度缩放到 1。长度为 1 的向量称为单位向量（unit vector）。

对于 2 维或 3 维的情况，上文提到，$\boldsymbol{y} - \boldsymbol{x}$经过平移，得到以$\boldsymbol{x}$和$\boldsymbol{y}$为邻边的三角形的第三边，$\boldsymbol{y} - \boldsymbol{x}$的模就是第三边的长度。三角形第三边的长度可用余弦公式求得。假如三角形的边长分别是a、b 和 c，a和b之间的夹角是θ，则c是：

$$c^2 = a^2 + b^2 - 2ab\cos\theta \tag{1.24}$$

当$\theta = \frac{\pi}{2}$时，$\cos\theta = 0$，式（1.24）就是毕达哥拉斯定理（勾股定理）。将三角形的边长替换为向量的模，则有：

$$\|\boldsymbol{y} - \boldsymbol{x}\|^2 = \|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - 2\|\boldsymbol{x}\|\|\boldsymbol{y}\|\cos\theta \tag{1.25}$$

根据模的定义将式（1.25）展开，消除等号两边相同项，得到：

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \|\boldsymbol{x}\| \|\boldsymbol{y}\| \cos\theta \tag{1.26}$$

式（1.26）说明：向量$\boldsymbol{x}$和$\boldsymbol{y}$的内积等于它们的长度之积再乘以它们之间夹角的余弦。在 2 维或 3 维的情况下，向量之间的夹角有直观的定义，这时运用余弦定理得到式（1.26）。在更高维的情况下，向量的夹角反过来由式（1.26）定义。如果向量$\boldsymbol{x}$和$\boldsymbol{y}$的内积为 0，则它们之间夹角的余弦是 0，即夹角θ是$\frac{\pi}{2}$，这时称$\boldsymbol{x}$与$\boldsymbol{y}$正交（orthogonal）。零向量与任意向量正交。若$\boldsymbol{x}$和$\boldsymbol{y}$都不是零向量，则它们的箭头互相垂直，如图 1-7 所示。

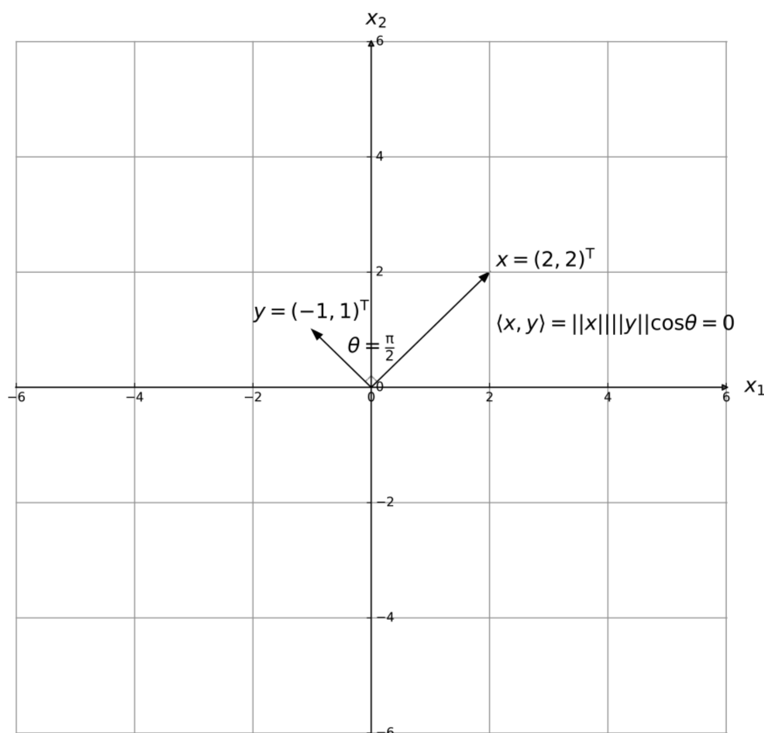


图 1-7　向量正交

如果$\boldsymbol{x}$和$\boldsymbol{y}$之间的夹角为θ，那么$\|\boldsymbol{x}\| \cos\theta$是$\boldsymbol{x}$向$\boldsymbol{y}$的投影的长度，如图 1-8 所示。投影长度等于$\frac{\langle x,y \rangle}{\|y\|}$，如果$\boldsymbol{y}$是单位向量，则$\boldsymbol{x}$向$\boldsymbol{y}$的投影的长度就等于$\langle \boldsymbol{x}, \boldsymbol{y} \rangle$。

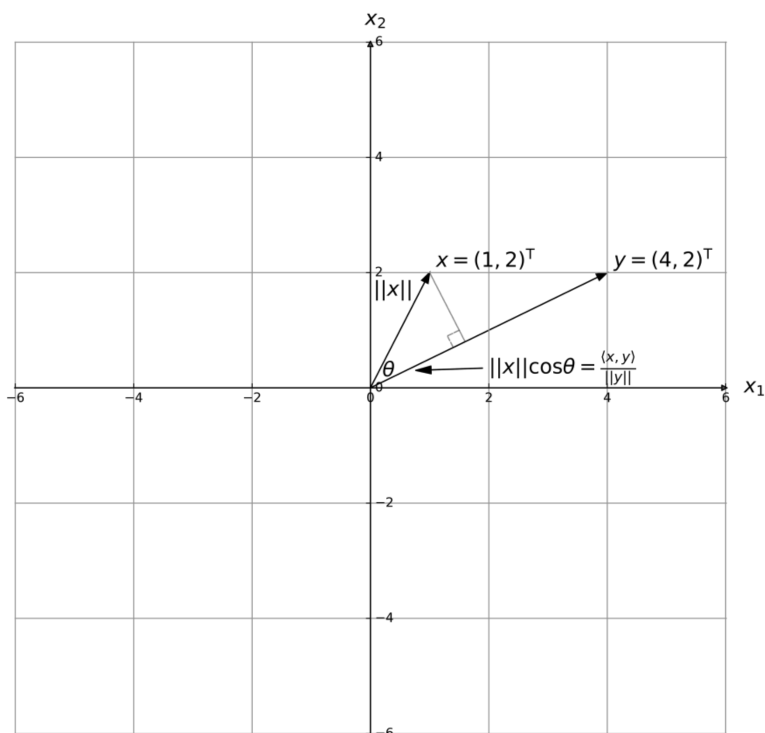


图 1-8 向量投影

如果把列向量看作$n \times 1$矩阵，则向量的内积可以用矩阵乘积表示（关于矩阵知识的详细讲解见 4.1 节）：

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{n} x_i y_i = (x_1, x_2, \cdots, x_n) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{y} \tag{1.27}$$

多数时候，我们用$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$表示向量$\boldsymbol{x}$和$\boldsymbol{y}$的内积。

1.2.4 线性空间、基与线性函数

我们可以将任何一个n维向量$\boldsymbol{x}$分解：

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + x_n \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \sum_{i=1}^{n} x_i \boldsymbol{e}^i \tag{1.28}$$

其中，$\boldsymbol{e}^i$是n维向量，它的第i维分量是 1，其余分量都是 0。上述公式称$\boldsymbol{x}$可被向量组$\boldsymbol{e}^i$（$i = 1, \cdots, n$）线性表出。任意n维向量都可以被这个向量组线性表出，但任何一个$\boldsymbol{e}^i$都无法被该向量组中的其他向量线性表出。如果一组向量中的任何一个都不能被组内其他向量线性表出，则称这组向量是线性独立的（linear independent）。如果一组向量不是线性独立的，则称它们线性相关。向量组$\boldsymbol{e}^i$（$i = 1, \cdots, n$）是线性独立的。

线性独立有一个等价定义。对于一组向量$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^n$，如果不存在一组不全为 0 的系数$w^1, w^2, \cdots, w^n$满足

$$\sum_{i=1}^{n} w^i \boldsymbol{v}^i = \boldsymbol{0} \tag{1.29}$$

则$\boldsymbol{v}^i$（$i = 1, \cdots, n$）是线性独立的。我们来证明这个结论。假设$\boldsymbol{v}^i$（$i = 1, \cdots, n$）是线性独立的，但是存在一组不全为 0 的系数满足式（1.29）。如果$w^j \neq 0$，则$\boldsymbol{v}^j$就可以被其他向量线性表出，即 $\boldsymbol{v}^j = \sum_{i \neq j} \frac{w^i}{w^j} v^i$，这就会产生矛盾。

如果不存在不全为 0 的系数满足式（1.29），但是$\boldsymbol{v}^i$（$i = 1, \cdots, n$）线性相关，那么其中某个向量$\boldsymbol{v}^j$可以被其他向量线性表出，即$\boldsymbol{v}^j = \sum_{i \neq j} w^i \boldsymbol{v}^i$。于是有$-\boldsymbol{v}^j + \sum_{i \neq j} w^i \boldsymbol{v}^i = \boldsymbol{0}$，也就是存在不全为 0 的系数满足式（1.29），这也会产生矛盾，所以两种线性独立的定义是等价的。

如果一组向量$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^s$可以被另一组向量$\boldsymbol{u}^1, \boldsymbol{u}^2, \cdots, \boldsymbol{u}^r$线性表出，并且$r < s$，那么向量组$\boldsymbol{v}^i$（$i = 1, \cdots, s$）是线性相关的。我们来证明这个结论，每一个向量$\boldsymbol{v}^i$都可以被$\boldsymbol{u}^j$（$j = 1, \cdots, r$）线性表出，即$\boldsymbol{v}^i = \sum_{j=1}^{r} a^{i,j} \boldsymbol{u}^j$。如果有一组系数$b^1, b^2, \cdots, b^s$满足

$$\sum_{i=1}^{s} b^i \boldsymbol{v}^i = \sum_{i=1}^{s} b^i \sum_{j=1}^{r} a^{i,j} \boldsymbol{u}^j = \sum_{j=1}^{r} \left(\sum_{i=1}^{s} a^{i,j} b^i\right) \boldsymbol{u}^j = \boldsymbol{0} \tag{1.30}$$

若要式（1.30）成立，只须对所有$j = 1, \cdots, r$，有$\sum_{i=1}^{s} a^{i,j} b^i = 0$即可，这是一个方程组：

$$\begin{cases} a^{1,1} b^1 + a^{2,1} b^2 + \cdots + a^{s,1} b^s = 0 \\ a^{1,2} b^1 + a^{2,2} b^2 + \cdots + a^{s,2} b^s = 0 \\ \quad \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \\ a^{1,r} b^1 + a^{2,r} b^2 + \cdots + a^{s,r} b^s = 0 \end{cases} \tag{1.31}$$

这个方程组有s个自变量、r个方程。因为$r < s$，方程数小于自变量数，所以这个方程组存在非全零解，即$\boldsymbol{v}^i$（$i = 1, \cdots, s$）线性相关。

如果一个向量集合满足“对于该集合中任意两个向量$\boldsymbol{x}$和$\boldsymbol{y}$，以及任意实数a和 b，向量$a\boldsymbol{x} + b\boldsymbol{y}$仍属于该集合”，则该集合是一个线性空间。全体$n$维向量的集合是一个线性空间，记为$\mathbb{R}^n$。如果一个线性空间中的所有向量都可以被$k$个线性独立的向量线性表出，则称这组向量为该线性空间的基（basis）。这个线性空间中任何多于k个向量的向量组必线性相关，因为它们可以被更少的

向量线性表出。任何少于k个向量的向量组必不能线性表出这个线性空间，否则那组k个向量的基就可以被更少的向量线性表出，这显然不可能。也就是说，凡是能线性表出该线性空间，且线性独立的向量组都只能有k个向量，不多不少。k称为该线性空间的维数。$\boldsymbol{e}^i$（$i=1,\cdots,n$）是$\mathbb{R}^n$的一组基，称为标准基。$\mathbb{R}^n$的维数是n。

两两正交的非零向量$\boldsymbol{v}^1,\boldsymbol{v}^2,\cdots,\boldsymbol{v}^k$一定是线性独立的，因为假如存在一组系数$w^1,w^2,\cdots,w^k$满足$\sum_{i=1}^k w^i\boldsymbol{v}^i=\boldsymbol{0}$，那么对于任何一个$j$，有：

$$(\boldsymbol{v}^j)^{\mathrm{T}}\sum_{i=1}^k w^i\boldsymbol{v}^i=\sum_{i=1}^k w^i(\boldsymbol{v}^j)^{\mathrm{T}}\boldsymbol{v}^i=w^j(\boldsymbol{v}^j)^{\mathrm{T}}\boldsymbol{v}^j=0 \quad (1.32)$$

因为$\boldsymbol{v}^j$是非零向量，它与自己的内积不为 0，所以必有$w^j=0$。这对所有j都成立，即$w^1,w^2,\cdots,w^k$都只能是 0，所以$\boldsymbol{v}^1,\boldsymbol{v}^2,\cdots,\boldsymbol{v}^k$是线性独立的。

现在介绍线性函数（linear function）。如果函数$f(\boldsymbol{x})$是线性的，则对于任意两个向量$\boldsymbol{x}$和$\boldsymbol{y}$与任意两个实数a和b，公式（1.33）成立：

$$f(a\boldsymbol{x}+b\boldsymbol{y})=af(\boldsymbol{x})+bf(\boldsymbol{y}) \quad (1.33)$$

如果$f(\boldsymbol{x})$是线性函数，则必然存在一个向量$\boldsymbol{w}$满足

$$f(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \quad (1.34)$$

也就是说，线性函数一定等于某个向量$\boldsymbol{w}$与输入向量$\boldsymbol{x}$的内积。这样构造向量$\boldsymbol{w}$：

$$\boldsymbol{w}=\begin{pmatrix} f(\boldsymbol{e}^1) \\ f(\boldsymbol{e}^2) \\ \vdots \\ f(\boldsymbol{e}^n) \end{pmatrix} \quad (1.35)$$

其中，$\boldsymbol{w}$的第i维分量是对$\boldsymbol{e}^i$施加$f(\boldsymbol{x})$后得到的值。由于$f(\boldsymbol{x})$是线性函数，有：

$$f(\boldsymbol{x})=f\left(\sum_{i=1}^n x_i\boldsymbol{e}^i\right)=\sum_{i=1}^n x_i f(\boldsymbol{e}^i)=\left(f(\boldsymbol{e}^1),f(\boldsymbol{e}^2),\cdots,f(\boldsymbol{e}^n)\right)\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \quad (1.36)$$

这就证明了式（1.34）的结论。线性函数的图像一定过原点，因为：

$$f(\boldsymbol{0})=f(0\times\boldsymbol{x})=0\times f(\boldsymbol{x})=0 \quad (1.37)$$

1.2.5　直线、超平面与仿射函数

对于向量$\boldsymbol{w} \neq \boldsymbol{0}$，令$\boldsymbol{x}$是任意与$\boldsymbol{w}$的内积为常数$C$的向量：

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \|\boldsymbol{w}\|\|\boldsymbol{x}\| \cos\theta = C \tag{1.38}$$

式（1.38）表明$\|\boldsymbol{x}\|\cos\theta = \frac{C}{\|\boldsymbol{w}\|}$，即$\boldsymbol{x}$向$\boldsymbol{w}$的投影的长度是常数$\frac{C}{\|\boldsymbol{w}\|}$。在 2 维的情况下，将$\boldsymbol{x}$看作平面上的一个点，则所有满足式（1.38）的点都位于垂直于$\boldsymbol{w}$的一条直线上，如图 1-9 所示。

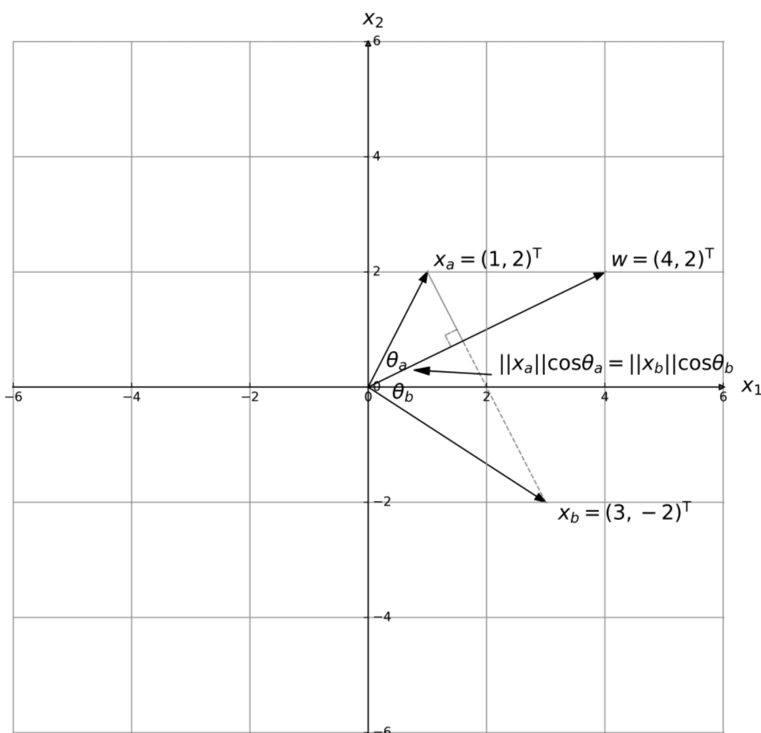


图 1-9　2 维空间中与非零向量内积相同的点构成垂直于该向量的直线

在 3 维空间中，所有与向量$\boldsymbol{w} \neq \boldsymbol{0}$的内积相同的点，构成垂直于$\boldsymbol{w}$的平面，如图 1-10 所示。在更高维空间中，这样的点构成垂直于$\boldsymbol{w}$的超平面（hyperplane）。

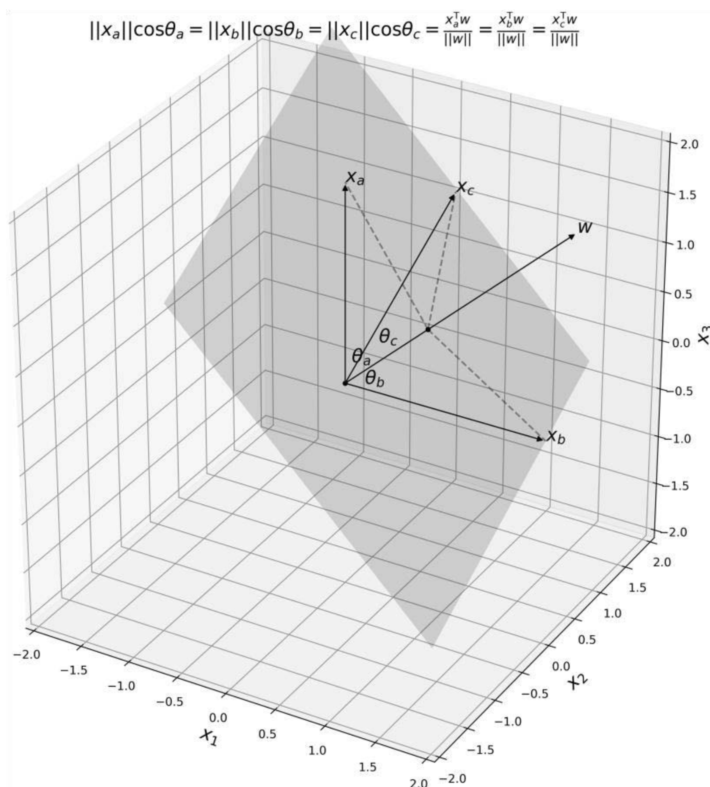


图 1-10　3 维空间中与非零向量内积相同的点构成垂直于该向量的平面

直线是 2 维空间中的超平面，平面是 3 维空间中的超平面。我们将直线、平面和更高维的超平面统称为超平面。$\boldsymbol{w}$称为超平面的法向量（norm）。对于任意非零实数k，向量$k\boldsymbol{w}$也是这个超平面的法向量，因为该超平面上所有向量与$k\boldsymbol{w}$的内积也相同：

$$(k\boldsymbol{w})^{\mathrm{T}}\boldsymbol{x} = \langle k\boldsymbol{w}, x\rangle = |k|\|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos\theta = |k|C \quad (1.39)$$

前文曾提到的仿射函数y，重新写在这里：

$$y = b + \sum_{i=1}^{n} w_i x_i \quad (1.40)$$

如果自变量是 2 维，将y移到等号另一侧，可得到：

$$w_1x_1 + w_2x_2 - y = (w_1, w_2, -1)\begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} = -b \quad (1.41)$$

根据式（1.41），在x_1x_2y三维空间中，所有满足式（1.40）的点与向量$\boldsymbol{w} = (w_1, w_2, -1)^{\mathrm{T}}$的

内积为常数$-b$。也就是说，仿射函数的图像是 3 维空间中的一张平面。该平面的法向量是$\boldsymbol{w}$。当$x_1 = x_2 = 0$时，y的值是b，b称为该平面的截距。由式（1.40）可以看出，仿射函数是线性函数加上一个常量b。

在 3 维情况下，若w_1和w_2的绝对值较大，则$\boldsymbol{w}$更贴近x_1x_2平面，以$\boldsymbol{w}$为法向量的平面接近竖立；若w_1和w_2的绝对值较小，则$\boldsymbol{w}$更贴近y轴，以$\boldsymbol{w}$为法向量的平面接近水平。平面的倾斜程度与w_1和w_2的绝对值大小有关，也就是与向量$(w_1, w_2)^{\mathrm{T}}$的模有关。$(w_1, w_2, 0)^{\mathrm{T}}$是法向量$\boldsymbol{w}$在$x_1x_2$平面上的投影，它的方向决定了平面的朝向。图 1-11 集中展示了这几个概念。

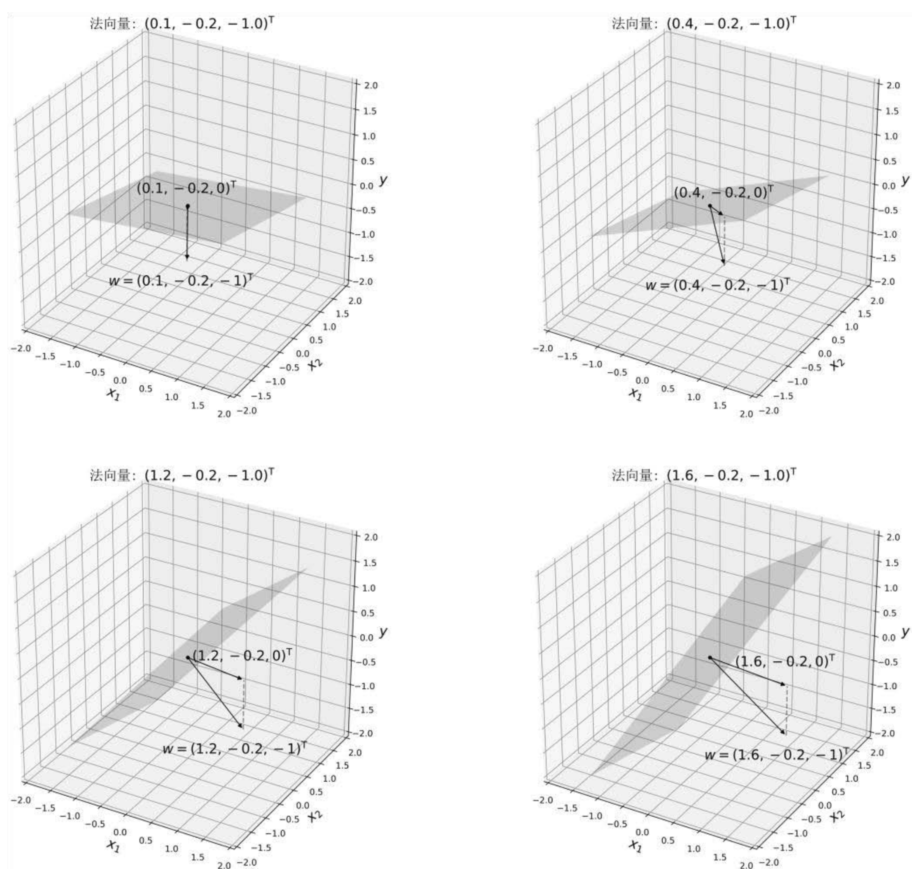


图 1-11　法向量决定平面的朝向和倾斜程度

仿射函数是一类最简单的函数，它的图像在自变量为 1 维的情况下是直线，在自变量为 2 维的情况下是平面，在更高维情况下是高维超平面。超平面在任意位置的性质都相同，例如 2 维平面任意位置的朝向和倾斜程度都相同。

1.3　从几何角度理解逻辑回归的能力和局限

我们回忆一下逻辑回归的表达式：

$$f(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-\left(b+\sum_{i=1}^{n}w_i x_i\right)}}=\frac{1}{1+\mathrm{e}^{-(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}} \tag{1.42}$$

$\boldsymbol{w}$是逻辑回归的权值向量，b是偏置。下面还是以 2 维为例，与权值向量$\boldsymbol{w}=(w_1,w_2)$垂直的直线上的向量和$\boldsymbol{w}$的内积都相同，所以对它们施加仿射函数$b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$的结果都相同，再施加 Logistic 函数的结果也都相同。逻辑回归先对输入向量施加仿射函数，这样就丢失了垂直于$\boldsymbol{w}$的方向上的信息。所以逻辑回归的输出只在$\boldsymbol{w}$的方向上有差异，在垂直于$\boldsymbol{w}$的方向上无差异。如果根据逻辑回归的输出是否大于阈值t来判断样本的类别：

$$\text{output}(\boldsymbol{x})=\begin{cases}0,\ f(\boldsymbol{x})<t\\1,\ f(\boldsymbol{x})\geqslant t\end{cases} \tag{1.43}$$

则只能得到垂直于$\boldsymbol{w}$的超平面分界面，这就是逻辑回归属于线性模型的原因。仿射函数是“罪魁祸首”，它选择了一个方向，忽略了其他方向上的信息，如图 1-12 所示。垂直于$\boldsymbol{w}$的那条虚线上的向量与$\boldsymbol{w}$的内积都相同，于是它们有相同的逻辑回归值。

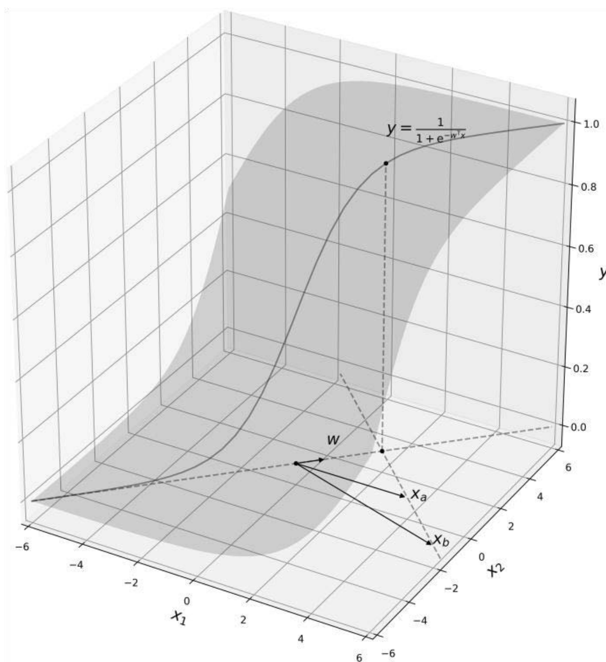


图 1-12　逻辑回归只能形成超平面分界面

神经元可以选用其他激活函数，但只要激活函数是单调的，就只能形成超平面分界面。采用单调激活函数的神经元无法处理异或（XOR）问题，因为异或问题是线性不可分的——无法用直线将正负样本点分隔开，如图 1-13 所示。

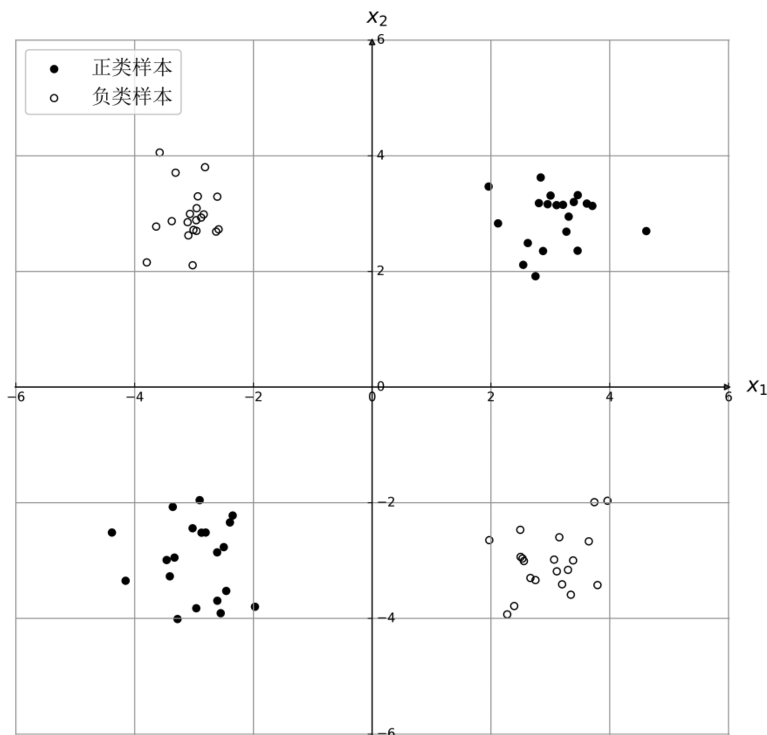


图 1-13　异或问题

解决异或问题，可以采用非单调激活函数，比如平方函数：

$$f(\boldsymbol{x}) = 0.02 \times (b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})^2 \tag{1.44}$$

$f(\boldsymbol{x})$沿$\boldsymbol{w}$的方向两头翘起，中间凹下。若$\boldsymbol{w}$取合适的方向并选择恰当的阈值，平方激活函数就可以解决异或问题，如图 1-14 所示。

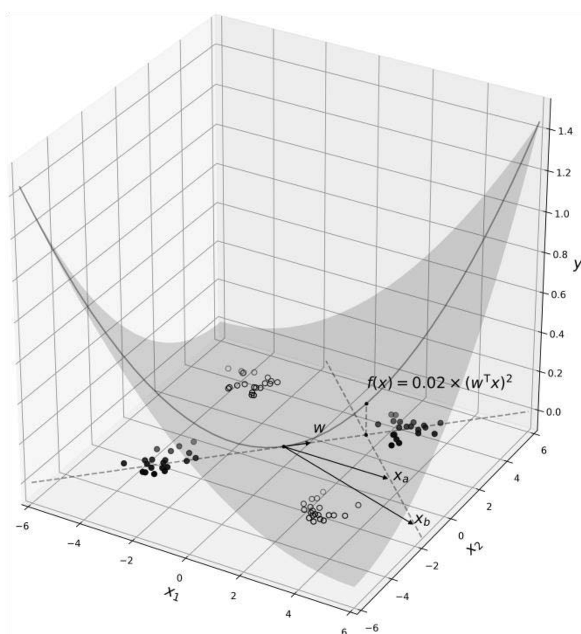


图 1-14 使用平方激活函数解决异或问题（偏置为 0）

但是式（1.44）的能力也仅限于用两条平行直线划分平面，就算使用正弦sin函数作激活函数，也只能形成无数条平行分界线。总之，垂直于权值向量方向的信息已经丢失了，神经元对于同心圆状分布的数据是无能为力的，如图 1-15 所示。

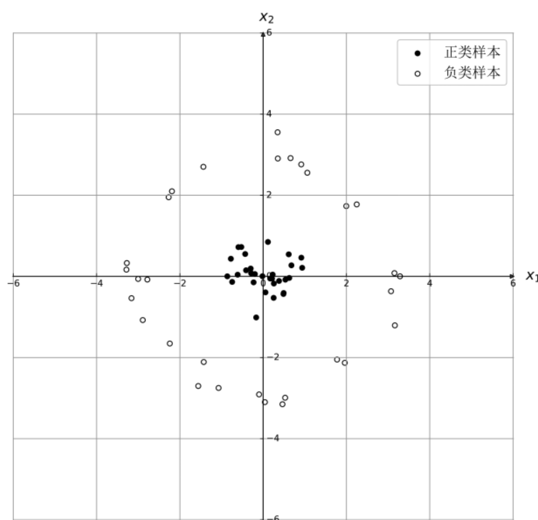


图 1-15 同心圆数据

同心圆数据的正类样本分布在中央，负类样本围绕在外围。沿着任意方向，都无法将两类样本分开。所以无论采用哪种激活函数，都无法分类同心圆数据。要想获得超越线性的分界能力，必须将多个神经元连接成网络，允许它们合作形成复杂分界面，这就是神经网络的思想。

1.4　实例：根据鸟类骨骼判断生态类群

本节中，我们将逻辑回归运用于一个实际问题：根据鸟类的前后肢骨骼测量指标来判断它所属的生态类群。全世界现存的鸟类有 9000 余种，可分为 8 个生态类群——游禽、涉禽、陆禽、猛禽、攀禽、鸣禽、走禽以及海洋性鸟类，不同生态类群的鸟类具有不同的生活习性。自然选择使鸟类的身体形态适应其生活环境。例如，涉禽有长长的脖颈和腿，这利于它们涉水捕鱼；猛禽有强壮的翅膀和有力的爪，这利于它们捕猎。

我们测量了 400 余个鸟类个体的前后肢骨骼，测量指标分别是：肱骨长度（huml）、肱骨宽度（humw）、尺骨长度（ulnal）、尺骨宽度（ulnaw）、股骨长度（feml）、股骨宽度（femw）、胫骨长度（tibl）、胫骨宽度（tibw）、跗跖骨长度（tarl）以及跗跖骨宽度（tarw）。这 10 个指标都是以毫米为单位的实数，精确到小数点后两位。括号中的内容是指标的名称，也就是特征名。最后一个字母 l 代表长度，w 代表宽度，之前是该骨骼拉丁文名称的前 3~4 个字母，具体见图 1-16。这 10 个特征能够反映鸟类翅膀和腿的绝对/相对长度和强壮程度。

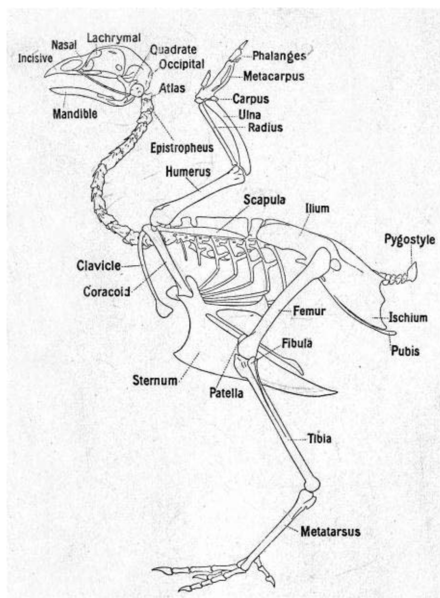


图 1-16　鸟类骨骼示意图

数据集的样本覆盖 8 个生态类群中的 6 个，它们分别是游禽（SW）、涉禽（W）、陆禽（T）、猛禽（R）、攀禽（P）以及鸣禽（SO），其中括号中的内容是我们为生态类群取的代号。数据集包含分属于 6 个类别的 413 个样本，每个样本包含 10 个数值特征和一个类别标签。随机选取一些例子，如表 1-1 所示。

表 1-1　数据集样例

huml	humw	ulnal	ulnaw	feml	femw	tibl	tibw	tarl	tarw	type
20.19	1.85	26.41	1.53	15.71	1.29	29.66	1.15	21.9	1.05	SO
85	5.07	93.17	4.21	38.33	2.57	67.32	2.73	40	2.53	W
107.41	5.69	110.5	5.32	40.62	3.15	81.86	3.71	44.62	2.95	SW
22.26	1.77	24.58	1.67	21.28	1.68	36.06	1.43	24.14	1.41	SO
48.19	3.44	43.71	3.01	50.46	3.6	94.86	3.45	55	4.57	W
100.38	5.52	113.24	5.04	44.72	4.07	82.87	3.69	54.8	3.25	W
23.27	2.15	27.62	2.16	21.86	1.64	35.55	1.51	23.23	1.32	SO
310	14.4	315	9.51	88.77	8.1	180	9.45	96.13	7.69	SW
29.26	2.66	27.63	2.23	27.87	2.03	38.9	1.81	25.22	1.62	P
186	9.83	152	8.76	56.02	7.02	185	8.07	90.8	4.59	SW

这 6 个类别的样本数量不均衡，具体如图 1-17 所示。

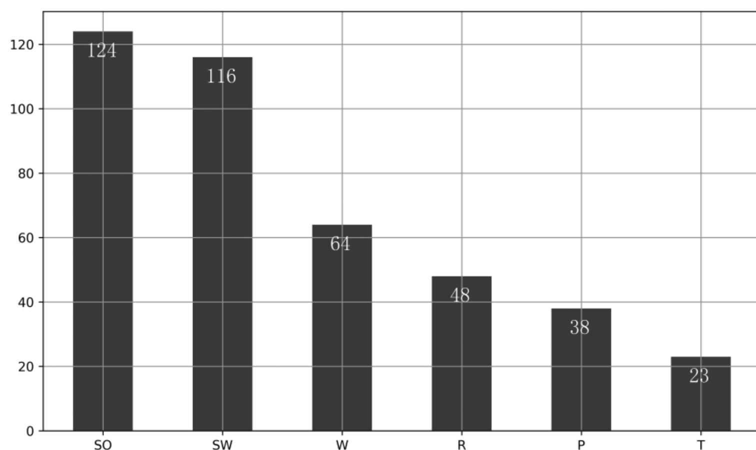


图 1-17　6 个类别（生态类群）的样本数量

第 6 章会介绍多分类逻辑回归，但现在我们介绍的逻辑回归只处理二分类问题，所以我们将 6 个类别归并为 2 个类别。首先，我们对数据做一些简单的分析。我们将 10 个特征每两个作为一对，画出二维散点图矩阵，如图 1-18 所示。

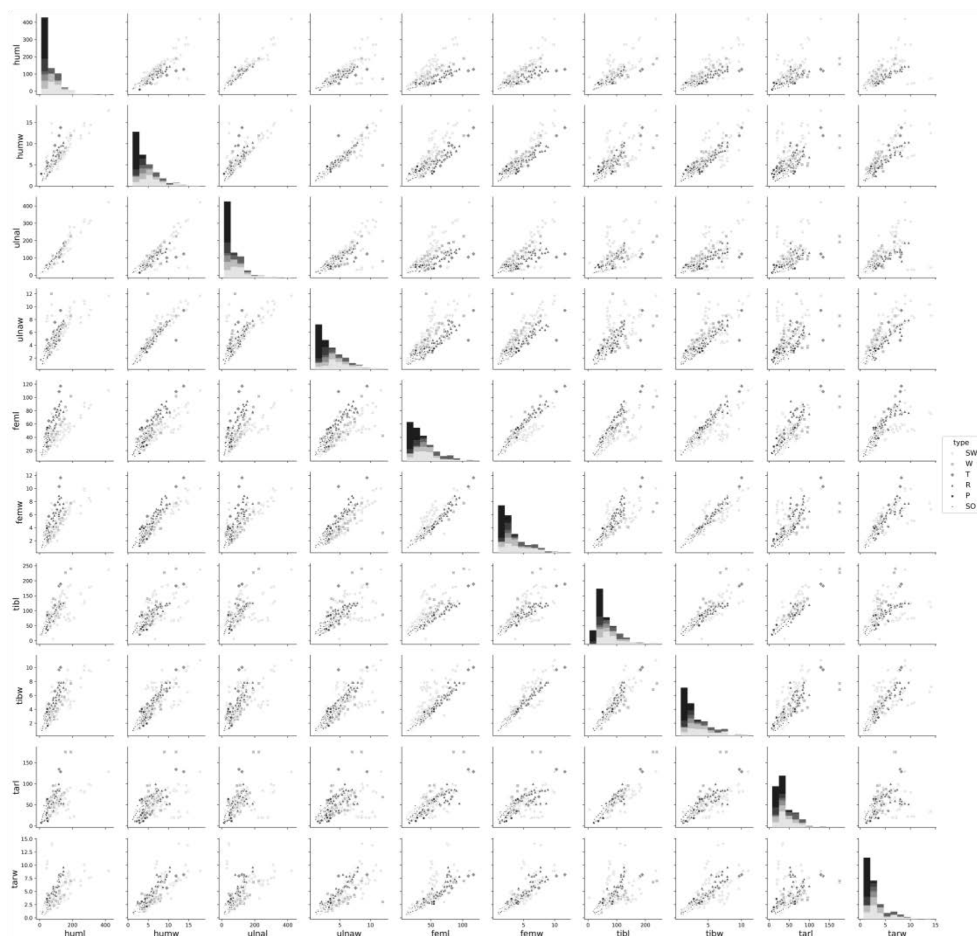


图 1-18　特征成对散点图

图 1-18 是一个 10 × 10 的阵列，每一个子图是行和列对应的两个特征的散点图，不同类别的样本用不同的标记画出。在对角线上，行和列对应同一个特征，散点图没有意义，所以对角线上的子图是该特征的分布直方图。各个类别有各自的直方图，以不同灰度显示。可以看出，特征与特征之间有较强的相关性，因为无论鸟的具体形态如何，大体型的鸟的所有骨骼都较长，小体型的鸟的所有骨骼都较短。

我们再用箱状图展示每个特征在每个类别上的分布，如图 1-19 所示。对于每个类别的每个特征，箱状图的“箱”中的三条横线分别对应特征的 25%、50%和 75%分位数，记为Q_1、Q_2和Q_3。令$\Delta Q = Q_3 - Q_1$，向上向下延伸的“须”分别达到$Q_3 + \Delta Q$和$Q_1 - \Delta Q$。在上下“须”之外的点，可以认为是异常点。箱状图大致勾画了特征的分布。

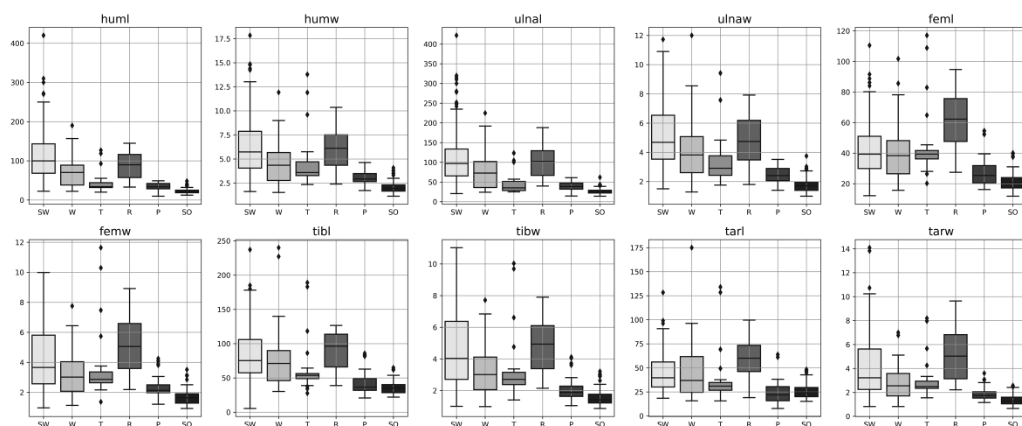


图 1-19　特征箱状图

从图 1-19 中可以看出，游禽（SW）、涉禽（W）和猛禽（R）这三个类别的各个测量值都较大，可以认为这三个类别是大型鸟，例如鹈鹕、火烈鸟和秃鹫。而陆禽（T）、攀禽（P）和鸣禽（SO）这三个类别是小型鸟，例如鸽子、鹦鹉和家燕。我们将 6 个类别归并为大型鸟和小型鸟这两个类别，以大型鸟为正类，这就构造了一个二分类问题。以不同标记表示大型鸟和小型鸟两个类别，5 个长度特征（名称以 l 为后缀的特征）的成对散点图如图 1-20 所示。

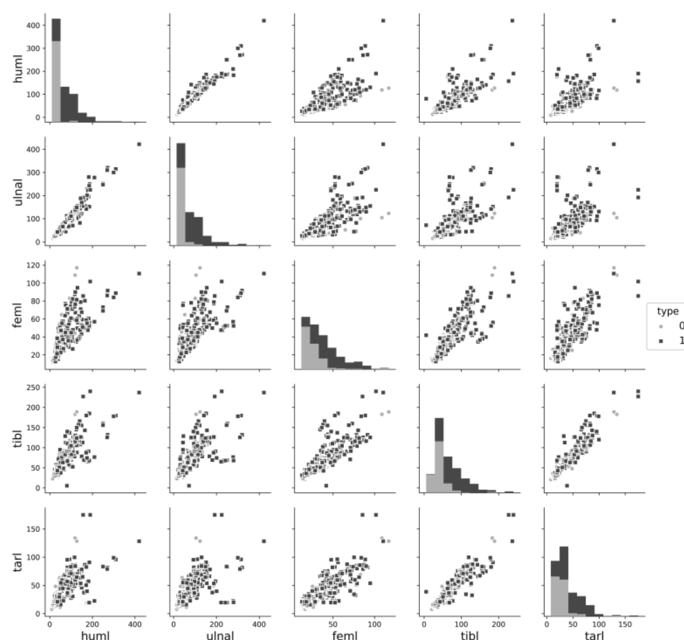


图 1-20　5 个长度特征区分两个类别的成对散点图

至此，我们构造了一个 413 个样本、10 个数值特征的二分类问题。对于这个问题，逻辑回归需要 10 个权值和 1 个偏置。在后续的章节中，我们将用原生 Python 实现逻辑回归和神经网络来对这个鸟类骨骼与生态类群问题进行建模。

1.5　小结

逻辑回归模型是一种线性模型，在神经网络的语境下，它是一个神经元。逻辑回归模型的分类能力的源泉在于它的权值向量。权值向量在空间中指示了一个方向，仿射函数值提取出数据沿该方向的差异。递增的激活函数随仿射函数值而变化，根据权值向量方向上的差异对不同样本产生不同输出，它可解释为概率。确定了概率阈值后，逻辑回归模型的分界面就是一个以权值向量为法向量的超平面。分界面将空间分成两个区域，不同区域中的样本被预测为不同的类别。

逻辑回归乃至线性模型的局限也就在于此：它们只能提取一个方向上的信息，垂直于权值向量方向上的信息丢失了。如果数据在所有方向上都有差异，就像同心圆数据那样，那么线性模型必然无法把握全部信息。后文我们会看到，只有将神经元连接成网络，才能克服这个局限。但在那之前，我们在下一章中先介绍如何根据目标调整逻辑回归模型的权值向量和偏置，即模型的训练问题。

第 2 章

模型评价与损失函数

逻辑回归模型对样本类别的预测取决于权值向量和偏置。想让模型能尽可能正确地预测样本的类别，就必须寻找合适的权值向量和偏置，它们是模型的参数。所谓“训练集”，是指包含真实类别标签的样本集合。“训练”则是指根据训练集寻找最优模型参数的过程。损失函数是模型参数的函数，它能够衡量模型参数的优劣。训练过程调整模型参数以最小化损失函数，将模型训练问题转化为函数优化问题。

我们首先介绍一些模型评价指标，这些指标从各种角度评价模型的预测效果。但是这些指标不能直接用作损失函数，因为无法根据它们调整模型参数。损失函数的种类很多，它们具有不同的特性，面向不同的问题。本章介绍分类问题最重要的损失函数——交叉熵损失，并从信息论和概率论两种角度解释交叉熵损失的含义。

通过本章，读者能够掌握模型训练的一般概念和模型评价的主要方法，并对交叉熵损失有较深刻的理解。至于优化损失函数的方法，则留待下两章讲解。本章虽是在逻辑回归的框架下进行讲解，但所有概念都可以用于其他机器学习模型。

2.1　训练集与测试集

给定权值向量$\boldsymbol{w}=(w_1,w_2,\cdots,w_n)^{\mathrm{T}}$和偏置$b$时，逻辑回归模型预测样本$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)^{\mathrm{T}}$属于正类的概率是：

$$f(x)=\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}x}} \quad (2.1)$$

其中，$\boldsymbol{w}$和b是模型的参数。“训练”（training）就是寻找参数$\boldsymbol{w}$和b，使模型尽可能正确地预测样本的类别。训练需要“训练集”（training set），它由一批带类别标签的样本组成。

样本的类别用一个实数y标识，例如用$y=1$标识样本属于正类，用$y=0$标识样本属于负类。

y值称为标签（label）。1/0 编码只是标签编码方法之一，编码还可以采用其他形式，后文会看到不同编码的用途。训练集是如式（2.2）描述的集合：

$$S = \{\boldsymbol{x}^i, y^i\}_{i=1}^{M} \tag{2.2}$$

其中，上标表示样本的编号，训练集S共包含M个样本。$\boldsymbol{x}^i \in \mathbb{R}^n$是由特征组成的向量，$y^i \in \{0,1\}$是样本的类别标签。为了客观评价模型的表现，必须有另一份带标签的样本集，这个样本集称为测试集（test set）。只有在测试集上评价模型，才能得到客观的指标。第 5 章会解释独立的测试集的意义。

2.2　分类模型的评价

逻辑回归模型判断样本$\boldsymbol{x}$属于正类的概率是$p(\boldsymbol{x} \in \mathrm{p})$。选择一个阈值$t$，当$p(\boldsymbol{x} \in \mathrm{p}) \geqslant t$时，预测样本$\boldsymbol{x}$属于正类，否则预测样本$\boldsymbol{x}$属于负类。

$$\hat{y} = \begin{cases} 0, & p(\boldsymbol{x} \in \mathrm{p}) < t \\ 1, & p(\boldsymbol{x} \in \mathrm{p}) \geqslant t \end{cases} \tag{2.3}$$

其中，$\hat{y}$是模型对样本的预测类别标签，同样使用 1/0 编码，顶上的“帽子”区分预测标签与真实标签。对测试集的所有样本计算概率$p(\boldsymbol{x} \in \mathrm{p})$，选择阈值后，根据式（2.3）就可以得出模型对每一个测试样本$\boldsymbol{x}^i$的预测标签$\hat{y}^i$。

2.2.1　混淆矩阵

有了测试集上的预测标签和真实标签，就可以得到混淆矩阵（confusion matrix），如表 2-1 所示。

表 2-1　二分类问题的混淆矩阵

	预测负类	预测正类
真实负类	TN	FP
真实正类	FN	TP

二分类问题的混淆矩阵是一个 2 × 2 矩阵，其他几种主要评价指标都可以从混淆矩阵计算得出。混淆矩阵的每一个元素分别如下所示。

- TN（true negative）：真实为负类，且模型预测为负类的样本数；
- FP（false positive）：真实为负类，但模型预测为正类的样本数（被错误地预测为正类）；
- FN（false negative）：真实为正类，但模型预测为负类的样本数（被错误地预测为负类）；
- TP（true positive）：真实为正类，且模型预测为正类的样本数。

2.2.2 正确率

正确率（accuracy）的计算公式为：

$$\text{accuracy}=\frac{\text{TN+TP}}{\text{TN+FP+FN+TP}} \tag{2.4}$$

正确率是混淆矩阵的对角线元素之和除以全体元素之和，它是模型预测正确的样本数与样本总数之比。正确率衡量模型预测的正确程度，但某些情况下它并非一个合适的评价指标。假如测试集中正类样本和负类样本的数量比为 99：1，那么如果模型将所有样本都预测为正类，正确率能达到 99%，但是该模型显然不是一个好模型。

2.2.3 查准率

查准率又称准确率（precision），其计算公式为：

$$\text{precision}_\text{p}=\frac{\text{TP}}{\text{TP+FP}} \tag{2.5}$$

其中，脚标表示这是正类的查准率，负类的查准率可以类似定义。$\text{precision}_\text{p}$是混淆矩阵右下角元素除以第二列元素之和，它是模型正确预测为正类的样本数与全部预测为正类的样本数（其中包括被错误地预测为正类的样本）之比。$\text{precision}_\text{p}$评价模型预测正类的准确程度，即$\text{precision}_\text{p}$越高，则模型的预测越可靠。

2.2.4 查全率

查全率又称召回率（recall），其计算公式为：

$$\text{recall}_\text{p}=\frac{\text{TP}}{\text{TP+FN}} \tag{2.6}$$

正类的查全率recall_p是混淆矩阵右下角元素除以第二行元素之和，它是模型预测为正类的样本数与全部正类样本数之比。类似地，我们也可以定义负类的查全率。recall_p评价模型对正类的召回情况：recall_p越高，说明模型能把更多的正类样本识别出来。recall_p又称真阳率（TPR，true positive rate），与之对应还有假阳率（FPR，false positive rate）：

$$\text{FPR}=\frac{\text{FP}}{\text{FP+TN}} \tag{2.7}$$

FPR是所有负类样本中被错误地预测为正类的比例。FPR越高，说明模型预测为正类的样本中混入越多的负类样本。

上述指标都衍生自混淆矩阵，它们基于模型的预测，而预测依赖概率阈值。如果阈值设得较

低，低门槛将导致更多的样本被预测为正类，$recall_p$会升高，但同时也会把更多负类样本错判为正类，从而抬高FPR，并降低$precision_p$。反之，若阈值设得较高，则$recall_p$和FPR会降低，但$precision_p$会升高。选择阈值是对两种相反倾向的权衡，需要根据具体问题的需求而定。

2.2.5　ROC 曲线

假阳率FPR和真阳率TPR这对指标随阈值变化同升同降，阈值低则两者都高，阈值高则两者都低。高TPR是我们愿意看到的，而高FPR则是希望避免的。我们希望在提高TPR的同时，不要大幅度地提高FPR。FPR和TPR随着阈值的变化可用 ROC 曲线（receiver operating characteristic curve）刻画，如图 2-1 所示。

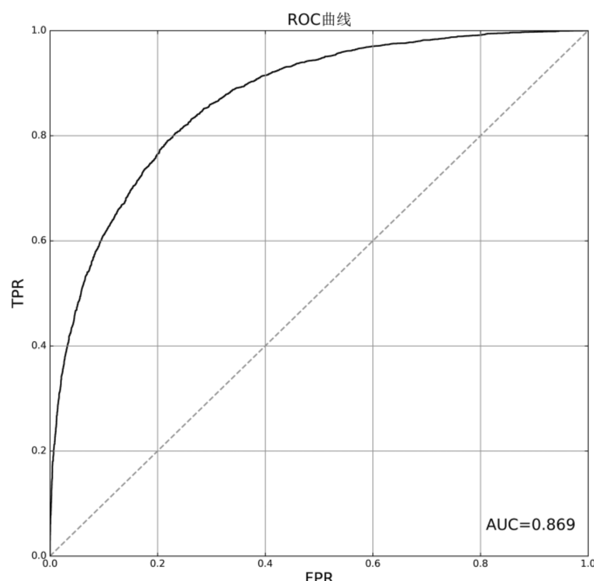


图 2-1　ROC 曲线

以FPR为横轴，以TPR为纵轴，将不同阈值对应的FPR和TPR以散点的形式画在坐标系中，得到一条向上拱起的曲线——ROC 曲线。ROC 曲线上拱得越高，说明在较低的FPR处有更高的TPR。ROC 曲线下的面积（area under curve，AUC）可以衡量模型的质量。高 AUC 意味着 ROC 曲线上拱得更高，模型的表现更优。AUC 不依赖阈值，是一个全面衡量模型质量的指标。

我们希望模型在测试集上有较优的表现，但我们不能直接用测试集上的评价指标来优化模型参数。模型参数的微小变化将导致概率的微小变化，当概率变化不足以跨越阈值时，模型对样本的预测不变，各种评价指标也就不变，这时没有任何信息能指导参数的优化。

2.3　损失函数

我们需要一个关于模型参数的可导函数，并且它能以某种方式衡量模型的效果。这种函数称为损失函数（loss function）。损失函数的值越小，则模型的预测效果越优。于是，模型训练问题就转化成了最小化损失函数的问题。损失函数有很多种，本节介绍分类问题中最常用的交叉熵（cross entropy）损失，并从信息论和贝叶斯两种视角阐释交叉熵损失的内涵。

2.3.1　K-L 散度与交叉熵

随机变量X有k种不同的取值：$x^1, x^2, \cdots, x^k$。记X的值取x^i的概率为$p(X=x^i)$，可简写作$p(x^i)$。若将X看作一个信号源，观察到$X=x^i$就相当于收到了一条信息。克劳德・香农定义了信息的信息量：

$$I(X=x^i) = \log\frac{1}{p(x^i)} = -\log p(x^i)\,,\quad i=1,\cdots,k \tag{2.8}$$

其中的对数可以 2 为底，也可取其他底，比如自然对数的底e。不同的底得到的信息量之间相差一个常系数。如果以 2 为底，信息量的单位是比特（bit）。$I(X=x^i)$是“$X=x^i$”这条信息的自信息量（self-information）。自信息量$I(X=x^i)$随着概率$p(x^i)$变化的图像如图 2-2 所示。当$p(x^i)$等于 1 时，自信息量$I(X=x^i)$为 0；当$p(x^i)$趋向于 0 时，$I(X=x^i)$趋向于正无穷。

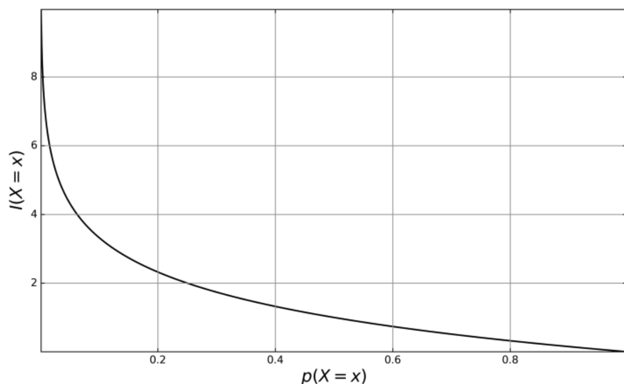


图 2-2　自信息量随概率变化的图像

自信息量定义背后的含义是：信息中的事件发生的概率越小，则信息量越大。假如有人告诉你：即将开奖的彩票中奖号码是 7006070，这条信息对你非常有用。假如有人告诉你：明天太阳照常升起，则这条信息几乎是无用的。你不用别人告诉也知道明天太阳肯定照常升起。前一条信息中的事件的概率极小，所以信息量很大；后一条信息中的事件的概率极大，所以信息量很小。

令信息源X取不同值$x^1, x^2, \cdots, x^k$的概率分别为$p(x^1), p(x^2), \cdots, p(x^k)$，定义信息源$X$的熵（entropy）为：

$$H(p) = \sum_{i=1}^{k} p(x^i) \log \frac{1}{p(x^i)} = -\sum_{i=1}^{k} p(x^i) \log p(x^i) \tag{2.9}$$

信息源由概率分布p描述，所以熵是p的函数，熵的概念来自热力学。$H(p)$又称作平均自信息，因为$H(p)$是将X的所有取值的自信息量以概率为权重取平均。换句话说$H(p)$是自信息量在分布p上的期望（exception）。式（2.9）针对的是离散型随机变量，对连续型随机变量可以用积分代替求和。

对于两个概率分布p和q，定义p与q的 K-L 散度（kullback-leibler divergence）是：

$$\mathrm{KLD}(p||q) = \sum_{i=1}^{k} p(x^i) \log \frac{p(x^i)}{q(x^i)} = -\sum_{i=1}^{k} p(x^i) \log q(x^i) - H(p) \tag{2.10}$$

K-L 散度是$\log\frac{p}{q}$在分布p上的期望，注意，$\mathrm{KLD}(p||q) \neq \mathrm{KLD}(q||p)$。如果对于所有$i$，都有$p(x^i) = q(x^i)$，即$\log\frac{p(x^i)}{q(x^i)} = 0$，则有$\mathrm{KLD}(p||q) = 0$。也就是说，两个相同分布$p$和$q$的 K-L 散度为 0，所以 K-L 散度可以用来衡量两个分布之间的差异程度。注意式（2.10）第二个等号后的第一项，将它定义为分布p和q的交叉熵（cross entropy）：

$$H(p,q) = -\sum_{i=1}^{k} p(x^i) \log q(x^i) \tag{2.11}$$

$H(p,q)$是$-\log q(x)$在分布p上的期望。根据式（2.10）和式（2.11），有：

$$H(p,q) = \mathrm{KLD}(p||q) + H(p) \tag{2.12}$$

分布p和q的交叉熵等于它们的 K-L 散度加上p的熵。如果分布p固定，则$H(p,q)$与$\mathrm{KLD}(p||q)$之间相差一个常数$H(p)$，于是$H(p,q)$也可以用来衡量分布p和q的差异程度：$H(p,q)$越小，则p和q越相似。对于一个训练样本$\{\boldsymbol{x}^i, y^i\}$，可以认为标签$y^i$给出了一个类别概率分布：

$$p(\boldsymbol{x}^i \in \mathrm{p}) = y^i,\quad p(\boldsymbol{x}^i \in \mathrm{n}) = 1 - y^i,\quad i = 1, \cdots, M \tag{2.13}$$

当$\boldsymbol{x}^i$属于正类时，$y^i = 1$，该分布就是$p(\boldsymbol{x}^i \in \mathrm{p}) = 1$且$p(\boldsymbol{x}^i \in \mathrm{n}) = 0$；当$\boldsymbol{x}^i$属于负类时，$y^i = 0$，该分布就是$p(\boldsymbol{x}^i \in \mathrm{p}) = 0$且$p(\boldsymbol{x}^i \in \mathrm{n}) = 1$。逻辑回归模型的输出也是一个分布$q$：

$$q(\boldsymbol{x}^i \in \mathrm{p}) = \frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}},\quad q(\boldsymbol{x}^i \in \mathrm{n}) = \frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}},\quad i = 1, \cdots, M \tag{2.14}$$

我们希望预测分布与标签给出的分布越相似越好，于是可以用标签给出的分布p和预测分布q的交叉熵作为训练样本$\{\boldsymbol{x}^i, y^i\}$的损失：

$$\text{loss}(\boldsymbol{w},b|\boldsymbol{x}^i,y^i)=-y^i\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}-(1-y^i)\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\,,\quad i=1,\cdots,M \qquad (2.15)$$

$\text{loss}(\boldsymbol{w},b|\boldsymbol{x}^i,y^i)$越小，则预测分布与标签给出的分布越相似。式（2.15）是一个训练样本的交叉熵，模型在整个训练集上的交叉熵就是所有训练样本的交叉熵的平均：

2

$$\text{loss}(\boldsymbol{w},b)=\frac{1}{M}\sum_{i=1}^{M}\left(-y^i\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}-(1-y^i)\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right) \qquad (2.16)$$

式（2.16）就是交叉熵损失，它基于特定训练集，并以参数$\boldsymbol{w}$和b为自变量。逻辑回归模型的训练就是寻找使$\text{loss}(\boldsymbol{w},b)$尽可能小的$\boldsymbol{w}$和$b$，这就将训练问题转化为函数优化问题。交叉熵损失与上一节介绍的各种评价指标之间没有直接的、显式的关系，但最小化交叉熵损失拉近了预测分布与训练样本给出的真实类别分布之间的“距离”。

2.3.2 最大似然估计

本节从概率的视角阐释交叉熵损失的内涵。令X和Y是两个离散型随机变量，贝叶斯公式（Bayes rule）是：

$$p(X=x|Y=y)=\frac{p(Y=y|X=x)p(X=x)}{p(Y=y)} \qquad (2.17)$$

$p(X=x|Y=y)$称为后验概率（posterior），它是观察到事件$Y=y$的前提下，事件$X=x$发生的概率。右边分子上的$p(X=x)$称为先验概率（prior），它是事件$X=x$发生的概率。分子上的$p(Y=y|X=x)$称为似然概率（likelihood），它是观察到事件$X=x$的前提下，事件$Y=y$发生的概率。分母$p(Y=y)$是事件$Y=y$发生的概率：

$$p(Y=y)=\sum_x p(Y=y,X=x)=\sum_x p(Y=y|X=x)p(X=x) \qquad (2.18)$$

由于对X的所有可能值求和，所以$p(Y=y)$与X的值无关。式（2.17）的证明很简单，将右边的分母乘到左边，根据条件概率的定义，等号两边都是$p(X=x,Y=y)$，即事件$X=x$和$Y=y$都发生的概率。

对于逻辑回归模型，事件X代表“模型的真实参数是$\boldsymbol{w}$和b”，事件Y代表“观察到训练集S”，代入贝叶斯公式：

$$p(\boldsymbol{w},b|S)=\frac{p(S|\boldsymbol{w},b)p(\boldsymbol{w},b)}{p(S)} \qquad (2.19)$$

式（2.19）中，在观察到训练集S的前提下，真实参数为$\boldsymbol{w}$和b的后验概率是$p(\boldsymbol{w},b|S)$。在真实参数为$\boldsymbol{w}$和b的前提下，观察到训练集S的似然概率是 $p(S|\boldsymbol{w},b)$。真实参数为$\boldsymbol{w}$和b的先验概率

是$p(\boldsymbol{w},b)$。观察到训练集S的概率是$p(S)$。这些概率满足贝叶斯公式。训练的目标，是寻找观察到训练集S的前提下，概率最大的参数值，也就是使后验概率$p(\boldsymbol{w},b|S)$最大的参数值：

$$\boldsymbol{w}^*,b^* = \mathrm{argmax}_{\boldsymbol{w},b}\ p(\boldsymbol{w},b|S) \tag{2.20}$$

假设先验概率$p(\boldsymbol{w},b)$与参数的取值无关，那么问题转化为寻找使似然概率最大的参数值：

$$\boldsymbol{w}^*,b^* = \mathrm{argmax}_{\boldsymbol{w},b}\ p(S|\boldsymbol{w},b) \tag{2.21}$$

其中，$\boldsymbol{w}^*,b^*$称为最大似然估计（maximum likelihood estimate，MLE）。对于一对训练样本和类别标签$\{\boldsymbol{x}^i,y^i\}$，y^i为 1 表示样本属于正类，为 0 表示样本属于负类。逻辑回归模型计算样本$\boldsymbol{x}^i$属于标签y^i所表示的类别的概率是：

$$p(y^i|\boldsymbol{w},b,\boldsymbol{x}^i) = \left(\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right)^{y^i} \cdot \left(\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right)^{1-y^i}, \quad i=1,\cdots,M \tag{2.22}$$

式（2.22）利用任何数的 0 次方都等于 1，根据y^i是 1 或 0，选择$p(\boldsymbol{x}^i \in \mathrm{p})$或$p(\boldsymbol{x}^i \in \mathrm{n})$。假设训练样本是独立的，有：

$$p(S|\boldsymbol{w},b) = \prod_{i=1}^{M} p(y^i|\boldsymbol{w},b,\boldsymbol{x}^i) \tag{2.23}$$

因为log是单调递增的，式（2.21）等价于寻找：

$$\boldsymbol{w}^*,b^* = \mathrm{argmax}_{\boldsymbol{w},b} \log p(S|\boldsymbol{w},b) \tag{2.24}$$

根据式（2.22）、式（2.23）和式（2.24），最大似然估计就是寻找$\boldsymbol{w}^*$和b^*，使式（2.25）最大化：

$$\log p(S|\boldsymbol{w},b) = \log\prod_{i=1}^{M} p(y^i|\boldsymbol{w},b,\boldsymbol{x}^i) = \sum_{i=1}^{M}\left(y^i\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}} + (1-y^i)\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right) \tag{2.25}$$

最大化$\log p(S|\boldsymbol{w},b)$等价于最小化它的相反数，于是最大似然估计就是寻找$\boldsymbol{w}^*$和$b^*$，使式（2.26）最小化：

$$-\log p(S|\boldsymbol{w},b) = -\sum_{i=1}^{M}\left(y^i\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}} + (1-y^i)\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right) \tag{2.26}$$

除了一个常系数$\frac{1}{M}$，式（2.26）和式（2.16）相同，所以最小化交叉熵（2.16）的$\boldsymbol{w}^*$和b^*正是最大似然估计。

最大似然估计使似然概率最大化，但其实我们真正想要的是最大化后验概率。在假设模型参数的先验概率与具体取值无关的前提下，此二者是等价的。在第 5 章中我们将看到，为损失函数加上“正则项”相当于为模型参数假设一个先验分布，最小化带“正则项”的损失函数等价于最大化后验概率，而正则化强度与先验分布的方差有关。

2.3.3 从几何角度理解交叉熵损失

第 1 章介绍过，逻辑回归只能形成超平面分界面（以下把 2 维直线、3 维平面以及更高维的超平面统称超平面）。如果以$p(\boldsymbol{x} \in \mathrm{p}) = 0.5$为阈值，则分界超平面的方程是：

$$p(\boldsymbol{x} \in \mathrm{p}) = \frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}} = 0.5 \qquad (2.27)$$

经过简单的计算，可知式（2.27）等价于：

$$b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = 0 \qquad (2.28)$$

满足式（2.28）的所有向量构成一个以$\boldsymbol{w}$为法向量的超平面$\mathbb{P}$，$\mathbb{P}$上的向量与$\boldsymbol{w}$的内积都相同，即$\mathbb{P}$上的向量向$\boldsymbol{w}$投影的长度都相同：

$$\frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}{\|\boldsymbol{w}\|} = -\frac{b}{\|\boldsymbol{w}\|}, \quad \boldsymbol{x} \in \mathbb{P} \qquad (2.29)$$

若某向量向$\boldsymbol{w}$投影的长度大于$-\frac{b}{\|\boldsymbol{w}\|}$，则它位于$\mathbb{P}$的一侧，该侧称为正侧；若某向量向$\boldsymbol{w}$投影的长度小于$-\frac{b}{\|\boldsymbol{w}\|}$，则它位于$\mathbb{P}$的另一侧，该侧称为负侧。$\boldsymbol{x}$的投影长度与$-\frac{b}{\|\boldsymbol{w}\|}$之差的绝对值$\frac{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}{\|\boldsymbol{w}\|}$是$\boldsymbol{x}$与超平面$\mathbb{P}$之间的距离，如图 2-3 所示。

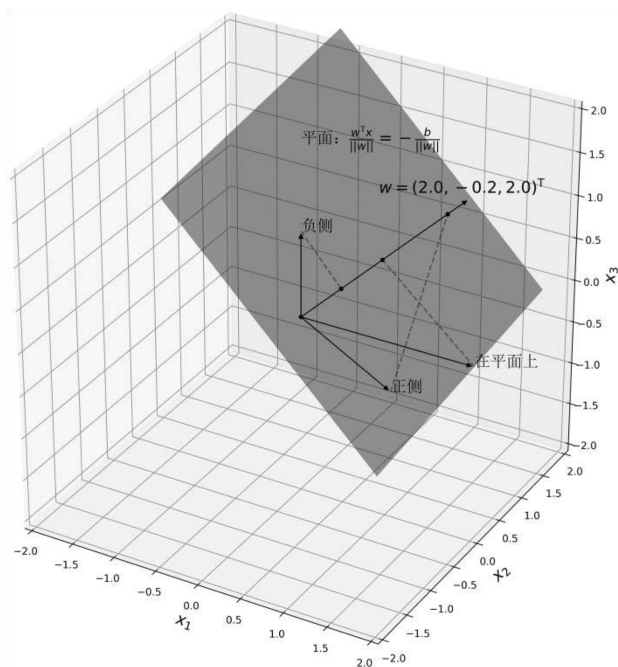


图 2-3 超平面将空间分为两侧

交叉熵损失衡量模型的预测分布与训练集标签给出的分布之间的差异程度，交叉熵越大，则这两个分布差异越大，所以训练的目标是最小化交叉熵。交叉熵损失是关于模型参数的一个非常复杂的函数，我们对这个函数做一些变形，目的是清晰地看出交叉熵损失究竟惩罚了模型什么样的行为。首先，我们用另一种方式编码样本的类别：

$$\tilde{y}^i=\begin{cases}1,\ \boldsymbol{x}^i\in \mathrm{p}\\-1,\ \boldsymbol{x}^i\in \mathrm{n}\end{cases},\quad i=1,\cdots,M \tag{2.30}$$

假设一个逻辑回归模型以 0.5 为阈值时，能正确预测全部训练样本，则对所有$\tilde{y}^i=1$的样本，有$p(\boldsymbol{x}^i\in \mathrm{p})>0.5$，于是$b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i>0$，$\boldsymbol{x}^i$位于$\mathbb{P}$的正侧，此时$\tilde{y}^i(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)>0$；对所有$\tilde{y}^i=-1$的样本，有$p(\boldsymbol{x}^i\in \mathrm{p})<0.5$，于是$b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i<0$，$\boldsymbol{x}^i$位于$\mathbb{P}$的负侧，此时仍有$\tilde{y}^i(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)>0$。

当$\tilde{y}^i(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)<0$，即$\tilde{y}^i$和$b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i$符号相反时，模型预测错误——正类样本位于$\mathbb{P}$的负侧，$p(\boldsymbol{x}^i\in \mathrm{p})<0.5$，负类样本位于$\mathbb{P}$的正侧，$p(\boldsymbol{x}^i\in \mathrm{p})>0.5$。损失函数应该对分错的情况施以惩罚。交叉熵损失式（2.16）中的y^i用 1/0 标识类别，对于所有i，y^i与$\tilde{y}^i$的关系是：

$$y^i=\frac{1+\tilde{y}^i}{2},\quad i=1,\cdots,M \tag{2.31}$$

将式（2.31）代入式（2.16），得到：

$$\mathrm{loss}(\boldsymbol{w},b)=\frac{1}{M}\sum_{i=1}^{M}\left(-\frac{1+\tilde{y}^i}{2}\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}-\frac{1-\tilde{y}^i}{2}\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right) \tag{2.32}$$

每一个训练样本$\{\boldsymbol{x}^i,\tilde{y}^i\}$对交叉熵损失的贡献是：

$$\begin{aligned}\mathrm{Loss}(\boldsymbol{w},b|\boldsymbol{x}^i,\tilde{y}^i)=-\frac{1+\tilde{y}^i}{2}\log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}-\frac{1-\tilde{y}^i}{2}\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}=\\-\frac{1}{2}\left(\tilde{y}^i\log\frac{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}+\log\frac{1}{\left(1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}\right)\left(1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}\right)}\right),\quad i=1,\cdots,M\end{aligned} \tag{2.33}$$

正类样本的$\tilde{y}^i=1$，于是式（2.33）变为：

$$\mathrm{loss}(\boldsymbol{w},b|\boldsymbol{x}^i,\tilde{y}^i)=-\frac{1}{2}\log\left(1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}\right)^{-2}=\log\left(1+\mathrm{e}^{-(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)}\right),\quad i=1,\cdots,M \tag{2.34}$$

负类样本的$\tilde{y}^i=-1$，于是式（2.33）变为：

$$\mathrm{loss}(\boldsymbol{w},b|\boldsymbol{x}^i,\tilde{y}^i)=-\frac{1}{2}\log\left(1+\mathrm{e}^{(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)}\right)^{-2}=\log\left(1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}\right),\quad i=1,\cdots,M \tag{2.35}$$

结合式（2.34）和式（2.35），得到：

$$\operatorname{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, \tilde{y}^i) = \log\left(1 + \mathrm{e}^{-\tilde{y}^i(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)}\right), \quad i = 1, \cdots, M \tag{2.36}$$

将$\operatorname{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, \tilde{y}^i)$视作$\tilde{y}^i(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)$的函数，其图像如图 2-4 所示。

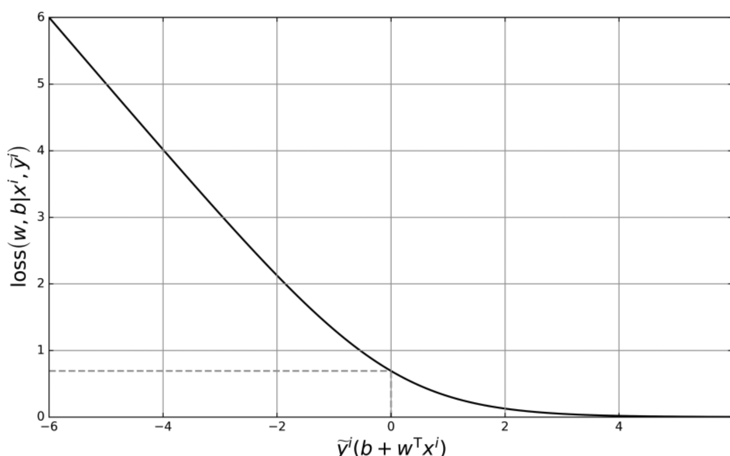


图 2-4　交叉熵损失作为$\tilde{y}^i\left(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i\right)$的函数的图像

如果对某一个训练样本$\{\boldsymbol{x}^i, \tilde{y}^i\}$，有$\tilde{y}^i(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i) > 0$，说明$\tilde{y}^i$与$b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i$同号，样本位于分界面正确的一侧，损失函数不应该惩罚这种情况。相反，若$\tilde{y}^i(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i) < 0$，说明$\tilde{y}^i$与$b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i$异号，样本位于分界面错误的一侧。而且$\left|b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i\right|$越大，$\boldsymbol{x}^i$在错误的一侧距离分界面越远，损失函数应该惩罚这种情况。从图 2-4 可见，交叉熵损失表现出恰当的行为：“安抚”正确的分类，惩罚错误的分类，并且对错误分类的惩罚力度随错误的程度呈线性增加。

2.4　小结

评价指标评价模型的预测效果，但众多的指标都只能反映模型效果的一个侧面，我们需要根据问题的具体需要选择不同的指标。本章介绍了最常用的几种模型评价指标，但它们无法作为优化模型参数的依据，因为它们不是关于参数的可导函数。什么是可导，为什么不可导就无法优化参数，留待后文讲解。

本章主要关注损失函数，损失函数是衡量模型预测效果的间接指标，例如交叉熵损失衡量模型预测分布与训练标签给出的分布之间的相似程度。从贝叶斯视角看，最小化交叉熵就是最大化观察到训练集的似然概率，这称为最大似然估计。从几何角度看，交叉熵损失惩罚位于分界面错误一侧的样本，距离分界面越远惩罚越重。损失函数将模型训练问题转化为函数优化问题，接下来的两章我们将介绍函数优化算法，它们是机器学习的核心数学基础之一。

第 3 章

梯度下降法

3

训练就是寻找使损失函数最小的模型参数，于是模型训练问题就成了函数优化问题。函数优化是应用数学的一个分支，是一个丰富的理论武器库。机器学习与函数优化联系紧密，特别是基于函数局部特性的迭代优化算法应用最为普遍。梯度下降法就是一种基于函数局部一阶特性的优化算法，它在神经网络和深度学习领域占统治地位。

要深入理解函数优化，必须具备多元微分的背景知识。本章首先回顾多元函数的梯度、方向导数、偏导数等概念。梯度包含函数在自变量空间某一点附近的一阶近似信息，或称线性近似。理解了梯度概念后，我们介绍梯度下降法。梯度下降法利用梯度确定函数值下降最快的方向，并沿该方向前进一段距离。算法迭代执行此步骤，使函数值不断下降，期待能够找到函数的全局最小点。

一阶近似是对原函数的最粗糙的近似。由于梯度下降法只利用了线性近似信息，所以它是短视的。本章阐述梯度下降法的一些问题，之后介绍几种对原始梯度下降法的改进。最后本章展示如何运用梯度下降法训练逻辑回归模型。通过本章，读者能够透彻理解梯度下降法的原理和局限。

3.1　多元函数的微分

微分刻画函数的局部线性近似，即用仿射函数近似原函数。仿射函数的图像是超平面，所以线性近似就是在局部用超平面近似原函数的图像。并不是所有函数在任意位置都能被仿射函数近似。若要这种近似能够成立，原函数必须满足一定条件，就是在该位置可微。前文介绍过，超平面的朝向和倾斜程度蕴含在它的法向量中。函数的局部近似超平面的法向量与函数在该位置的梯度有关。

多元函数的自变量在某点可以沿无数方向运动，函数值在每个方向上都有不同的变化率。多元函数沿某一方向的变化率称为方向导数。各个方向的方向导数也都蕴含在梯度之中。若要寻找函数值下降最快的方向，就要用到梯度。本节深入讲解这些概念。

3.1.1 梯度

首先回忆一下一元函数$f(x)$的可导性及其导数$f'(x)$的定义：

$$f'(x)=\frac{\mathrm{d}f}{\mathrm{d}x}(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h} \tag{3.1}$$

如果式（3.1）的极限存在，则称$f(x)$在x可导，导数是$f'(x)$。h是一个变化量。在$f(x)$的图像中用一个线段连接$\left(x,f(x)\right)^{\mathrm{T}}$和$\left(x+h,f(x+h)\right)^{\mathrm{T}}$两点，这个线段称为割线。式（3.1）极限中的商$\frac{f(x+h)-f(x)}{h}$是割线的斜率。随着$h$趋近于 0，割线趋近于$f(x)$在$x$的切线（tangent line）。函数在某一点割线斜率的极限就是函数在该点的导数——切线的斜率，如图 3-1 所示。

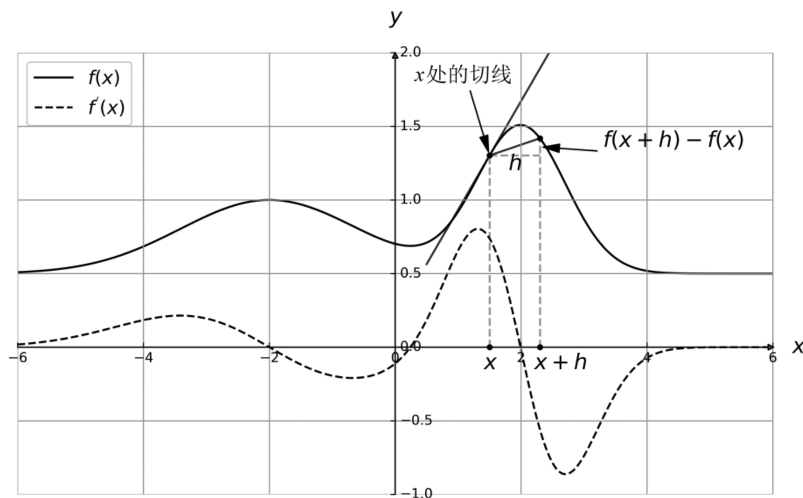


图 3-1 一元函数的割线、切线和导数

自变量从x变化到$x+h$时，割线斜率$\frac{f(x+h)-f(x)}{h}$也是函数值的平均变化率。导数$f'(x)$是平均变化率的极限——函数值在x的瞬时变化率。在一元情况下，自变量只能沿着x轴前后运动，这时可以像式（3.1）那样用瞬时变化率定义导数。但多元函数的自变量是向量，可以沿无数方向运动，那么如何定义多元函数的导数呢？

一元函数的可导性还有另一种等价定义：若在x附近能用直线近似$f(x)$的图像，则称$f(x)$在x可导。后面我们将这种可导定义扩展到多元函数。但首先我们要说明"能用直线近似"是什么意思，并证明在一元情况下这两种可导定义是等价的。假设$f(x)$在x可导，构造一个关于h的仿射函数：

$$g(h)=f(x)+hf'(x) \tag{3.2}$$

将$f(x+h)$写成一个关于h的仿射函数再加余项的形式：

$$f(x+h)=f(x)+hf'(x)+\mathcal{R}(h) \tag{3.3}$$

若函数在某点可导，则它在该点连续。于是$f(x)$在x连续，即关于h的函数$f(x+h)$在$h=0$连续。仿射函数$g(h)$也是连续的。余项$\mathcal{R}(h)$是两个连续函数的差，所以它也在$h=0$连续，即$\lim_{h\to 0}\mathcal{R}(h)=\mathcal{R}(0)=0$。另外，根据式（3.1）有：

$$\lim_{h\to 0}\left|\frac{\mathcal{R}(h)}{h}\right|=\lim_{h\to 0}\left|\frac{f(x+h)-f(x)}{h}-f'(x)\right|=0 \tag{3.4}$$

式（3.4）等价于：

$$\lim_{h\to 0}\frac{\mathcal{R}(h)}{|h|}=0 \tag{3.5}$$

所以，当h趋近于 0 时，$\mathcal{R}(h)$和$\left|\frac{\mathcal{R}(h)}{h}\right|$都趋近于 0。这说明随着变化量$h$消失，$\mathcal{R}(h)$也消失，而且消失得比$h$还快。这种情况称$\mathcal{R}(h)$是$h$的高阶无穷小。对式（3.3）做一个变量替换，令$x'=x+h$，得到：

$$f(x')=f(x)+f'(x)(x'-x)+\mathcal{R}(h)\approx f(x)+f'(x)(x'-x) \tag{3.6}$$

式（3.6）的含义是：函数在x附近可被一个仿射函数近似，近似误差是x'与x之间距离$|x'-x|$的高阶无穷小。该仿射函数的图像是经过点$\left(x,f(x)\right)^{\mathrm{T}}$的直线，即$f(x)$在$x$的切线。切线在$x$点与$f(x)$的图像重合，在其他点与$f(x)$的图像有差距，差距是自变量与$x$之间距离的高阶无穷小。这就是函数的图像在$x$点附近可被直线近似的含义，如图 3-2 所示。

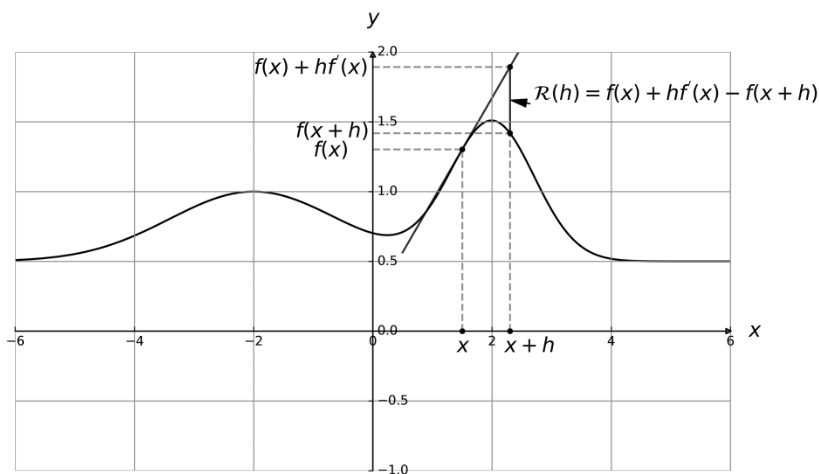


图 3-2　可导函数在某点附近可被其切线近似

现在反过来证明：如果$f(x)$的图像在x点附近可被直线近似，则$f(x)$在x可导且导数是该直线的斜率。如果$f(x)$在x附近的值$f(x+h)$可以写成$f(x+h)=f(x)+ha+\mathcal{R}(h)$，其中$\mathcal{R}(h)$是$h$的高阶无穷小，那么有：

$$\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}=\lim_{h\to 0}\frac{ha+\mathcal{R}(h)}{h}=a+\lim_{h\to 0}\frac{\mathcal{R}(h)}{h}=a \tag{3.7}$$

式（3.7）说明$f(x)$在x可导，导数$f'(x)=a$。这就证明了两种可导定义是等价的。以“可被仿射函数近似”作为可导的定义，就可将可导性扩展到多元函数$f(\boldsymbol{x})$。如果$f(\boldsymbol{x})$在$\boldsymbol{x}$附近的值$f(\boldsymbol{x}+\boldsymbol{h})$可以被某个仿射函数近似，即存在记为$\nabla f(\boldsymbol{x})$的向量，满足：

$$f(\boldsymbol{x}+\boldsymbol{h})=f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h}+\mathcal{R}(\boldsymbol{h}) \tag{3.8}$$

其中，余项$\mathcal{R}(\boldsymbol{h})$是$\boldsymbol{h}$的长度$\|\boldsymbol{h}\|$的高阶无穷小：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|}=0 \tag{3.9}$$

则称多元函数$f(\boldsymbol{x})$在$\boldsymbol{x}$可导。式（3.8）中的向量$\nabla f(\boldsymbol{x})$称为$f(\boldsymbol{x})$在$\boldsymbol{x}$的梯度（gradient）。对式（3.8）做变量替换，令$\boldsymbol{x}'=\boldsymbol{x}+\boldsymbol{h}$，得到：

$$f(\boldsymbol{x}')=f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{x}'-\boldsymbol{x})+\mathcal{R}(\boldsymbol{x}'-\boldsymbol{x})\approx f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{x}'-\boldsymbol{x}) \tag{3.10}$$

式（3.10）的含义是：函数在$\boldsymbol{x}$附近可被一个仿射函数近似，近似误差是$\boldsymbol{x}'$与$\boldsymbol{x}$之间距离$\|\boldsymbol{x}'-\boldsymbol{x}\|$的高阶无穷小。这个仿射函数的图像是经过点$\left(\boldsymbol{x}^{\mathrm{T}},f(\boldsymbol{x})\right)^{\mathrm{T}}$的超平面，即函数在$\boldsymbol{x}$的切平面（tangent hyperplane）。切平面的法向量是$(\nabla f(\boldsymbol{x})^{\mathrm{T}},-1)^{\mathrm{T}}$。于是梯度的方向决定切平面的朝向，梯度的长度（模）决定切平面的倾斜程度，如图 3-3 所示。

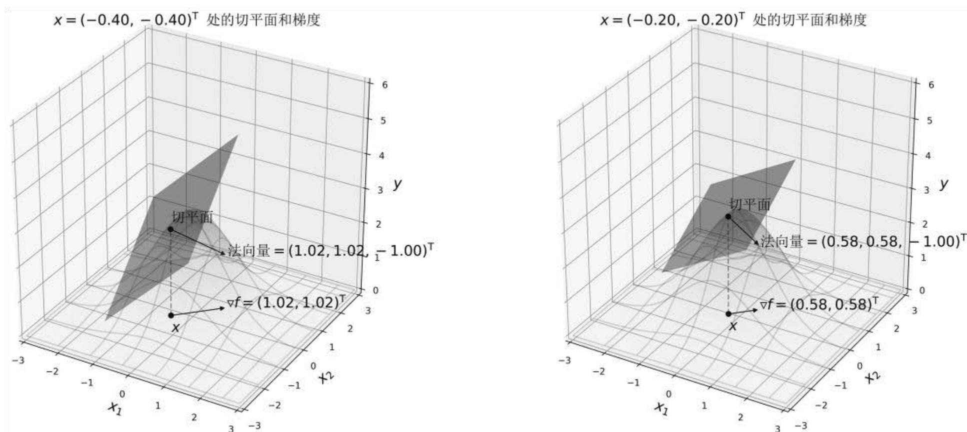


图 3-3　多元函数的切平面和梯度

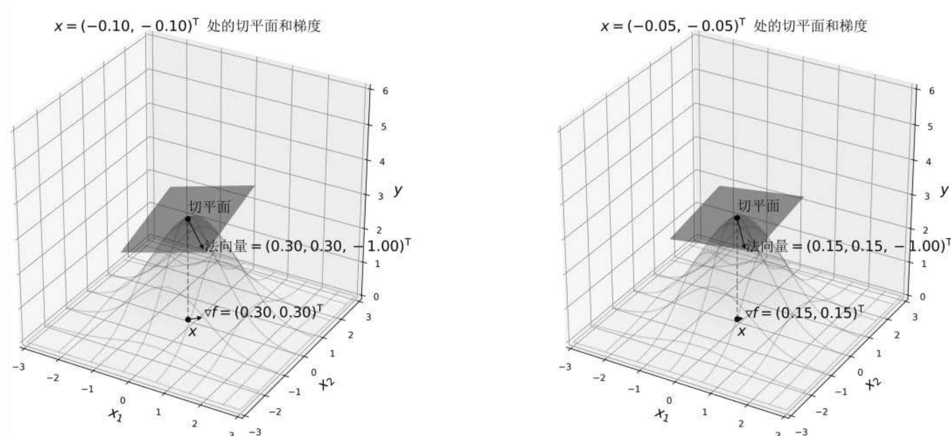


图 3-3　（续）

3.1.2　方向导数

如果$f(\boldsymbol{x})$在$\boldsymbol{x}$可导，如何定义$f(\boldsymbol{x})$在$\boldsymbol{x}$沿各个方向的变化率？可以指定一条经过$\boldsymbol{x}$的直线，然后讨论当自变量沿着这条直线运动时函数值的变化率。经过$\boldsymbol{x}$的直线可定义为：

$$l(t) = \boldsymbol{x} + t\boldsymbol{d},\quad t \in \mathbb{R},\quad \|\boldsymbol{d}\| = 1 \tag{3.11}$$

其中，$\boldsymbol{d}$是单位向量，它决定了直线的走向。t决定了$\boldsymbol{x} + t\boldsymbol{d}$离$\boldsymbol{x}$的距离，$l(0) = \boldsymbol{x}$。接下来在直线$l(t)$的基础上定义一个复合函数$(f\oplus l)(t)$：

$$(f\oplus l)(t) = f\big(l(t)\big) = f(\boldsymbol{x} + t\boldsymbol{d}) \tag{3.12}$$

$(f\oplus l)(t)$是自变量为t的一元函数。根据式（3.1），$(f\oplus l)(t)$在 0 的导数是：

$$\frac{\mathrm{d}(f\oplus l)}{\mathrm{d}t}(0) = \lim_{h\to 0}\frac{f\big(l(h)\big) - f\big(l(0)\big)}{h} = \lim_{h\to 0}\frac{f(\boldsymbol{x} + h\boldsymbol{d}) - f(\boldsymbol{x})}{h} \tag{3.13}$$

式（3.13）就是$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\boldsymbol{d}$的方向导数（directional derivative），用$\nabla_{\boldsymbol{d}} f(\boldsymbol{x})$表示。$\boldsymbol{x}$和单位向量$\boldsymbol{d}$定义了一个坐标轴，$\boldsymbol{x}$是该坐标轴的原点。坐标轴的走向沿着$\boldsymbol{d}$，以$\boldsymbol{d}$的方向为正方向。如果限制自变量只能在这条坐标轴上变化，这就相当于定义了一个一元函数。$\nabla_{\boldsymbol{d}} f(\boldsymbol{x})$是这个一元函数在原点的导数，如图 3-4 所示。

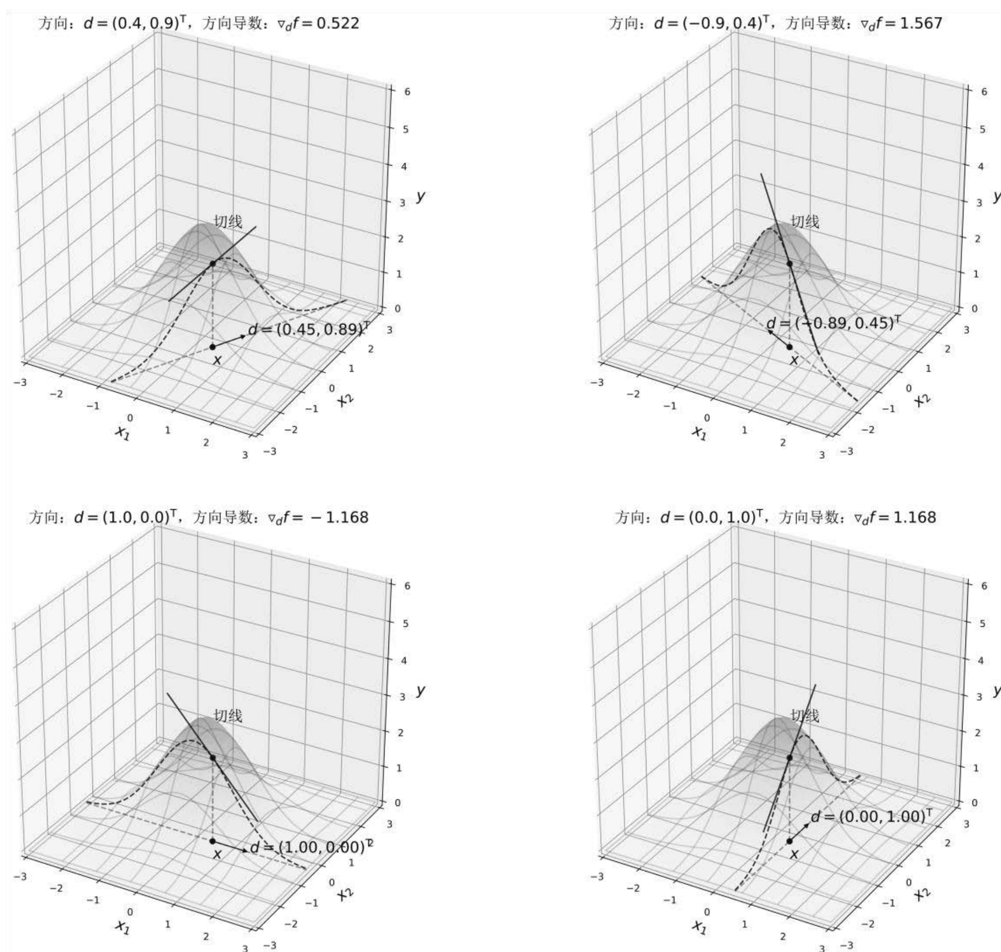


图 3-4　方向导数

$\boldsymbol{d}$必须是单位向量，否则$(f\oplus l)(t)$的t就不再是自变量沿新坐标轴运动的距离。$-\boldsymbol{d}$确定了一个方向相反的坐标轴，函数的变化率相反，即$\nabla_{-\boldsymbol{d}}f(\boldsymbol{x})=-\nabla_{\boldsymbol{d}}f(\boldsymbol{x})$，这容易验证。$f(\boldsymbol{x})$在$\boldsymbol{x}$沿各个方向的方向导数都蕴含在梯度$\nabla f(\boldsymbol{x})$中。因为$f(\boldsymbol{x})$在$\boldsymbol{x}$可导，根据式（3.8）有：

$$(f\oplus l)(h)=f(\boldsymbol{x}+h\boldsymbol{d})=f(\boldsymbol{x})+h\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+\mathcal{R}(h\boldsymbol{d}) \tag{3.14}$$

其中，$\mathcal{R}(h\boldsymbol{d})$是变化量$h\boldsymbol{d}$的高阶无穷小，即：

$$\lim_{\|h\boldsymbol{d}\|\to 0}\frac{\mathcal{R}(h\boldsymbol{d})}{\|h\boldsymbol{d}\|}=0 \tag{3.15}$$

因为$\|h\boldsymbol{d}\|=|h|\|\boldsymbol{d}\|=|h|$，所以当$h$趋近于 0 时，$\|h\boldsymbol{d}\|$也趋近于 0。这时有：

$$\lim_{h\to 0}\frac{\mathcal{R}(h\boldsymbol{d})}{|h|}=\lim_{h\to 0}\frac{\mathcal{R}(h\boldsymbol{d})}{\|h\boldsymbol{d}\|}=\lim_{\|h\boldsymbol{d}\|\to 0}\frac{\mathcal{R}(h\boldsymbol{d})}{\|h\boldsymbol{d}\|}=0 \tag{3.16}$$

所以$\mathcal{R}(h\boldsymbol{d})$也是$h$的高阶无穷小。另外，因为$f(\boldsymbol{x})=(f\oplus l)(0)$，所以式(3.14)表明$(f\oplus l)(t)$在 0 的导数是$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$，即$\nabla_{\boldsymbol{d}}f(\boldsymbol{x})=\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$。于是我们有：

$$\nabla_{\boldsymbol{d}}f(\boldsymbol{x})=\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}=\|\nabla f(\boldsymbol{x})\|\cdot\|\boldsymbol{d}\|\cos\theta=\|\nabla f(\boldsymbol{x})\|\cos\theta \tag{3.17}$$

其中，θ是$\nabla f(\boldsymbol{x})$与$\boldsymbol{d}$之间的夹角。由式（3.17）可知，$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\boldsymbol{d}$的方向导数等于$f(\boldsymbol{x})$在$\boldsymbol{x}$的梯度$\nabla f(\boldsymbol{x})$向$\boldsymbol{d}$的投影长度。当$\boldsymbol{d}$与$\nabla f(\boldsymbol{x})$同方向（$\theta=0$）时，$\nabla_{\boldsymbol{d}}f(\boldsymbol{x})$最大。所以$\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}$是$f(\boldsymbol{x})$变化率最大的方向，其变化率是$\|\nabla f(\boldsymbol{x})\|\geqslant 0$。沿着$-\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}$方向（$\theta=\pi$）的方向导数最小，等于$-\|\nabla f(\boldsymbol{x})\|\leqslant 0$。所以，梯度反方向$-\nabla f(\boldsymbol{x})$是$f(\boldsymbol{x})$变化率最小的方向，即函数值下降最快的方向。

在 2 维的情况下，可以用切平面描述梯度$\nabla f(\boldsymbol{x})$与方向导数$\nabla_{\boldsymbol{d}}f(\boldsymbol{x})$的关系。切平面的法向量是$(\nabla f(\boldsymbol{x})^{\mathrm{T}},-1)^{\mathrm{T}}$，第 3 维$-1$说明法向量指向下方。法向量在$x_1x_2$平面的投影是$\nabla f(\boldsymbol{x})$，它指向切平面的上坡方向，$-\nabla f(\boldsymbol{x})$指向切平面的下坡方向，如图 3-3 所示。

自变量沿任意方向$\boldsymbol{d}$的运动可分解为两个分量：平行于$\nabla f(\boldsymbol{x})$的分量和垂直于$\nabla f(\boldsymbol{x})$的分量。垂直于$\nabla f(\boldsymbol{x})$的分量上，$\nabla_{\boldsymbol{d}}f(\boldsymbol{x})=0$。自变量沿$\boldsymbol{d}$的运动一部分摊在了垂直分量上，这部分运动不会导致$f(\boldsymbol{x})$变化。只有在平行于$\nabla f(\boldsymbol{x})$的分量上的运动才导致$f(\boldsymbol{x})$变化。所以$f(\boldsymbol{x})$沿$\boldsymbol{d}$的变化率就打了折扣，折扣系数是平行于$\nabla f(\boldsymbol{x})$的分量所占的“份额”——$\cos\theta$，如图 3-5 所示。

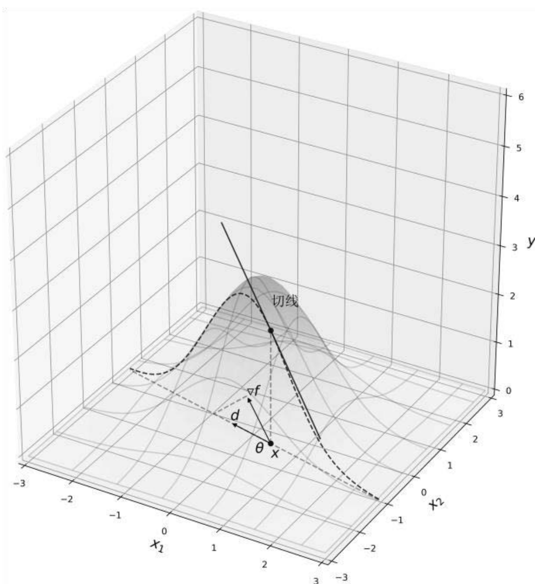


图 3-5　梯度与方向导数

3.1.3 偏导数

$f(\boldsymbol{x})$在$\boldsymbol{x}$点对其第i分量x_i的偏导数是把其他分量x_j（$j \neq i$），当作常数时$f(\boldsymbol{x})$对x_i的导数。这时候将$f(\boldsymbol{x})$看作关于x_i的一元函数。根据导数的定义：

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{e}^i) - f(\boldsymbol{x})}{h} \quad (3.18)$$

其中，$\boldsymbol{x} + h\boldsymbol{e}^i$保持$x_j$($j \neq i$)不变，只让x_i发生变化，变化量是h。$f(\boldsymbol{x})$有n个偏导数：$\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}$。根据式（3.13），有：

$$\nabla_{\boldsymbol{e}^i} f(\boldsymbol{x}) = \frac{\partial f}{\partial x_i}(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{e}^i) - f(\boldsymbol{x})}{h} \quad (3.19)$$

$f(\boldsymbol{x})$对x_i的偏导数就是$f(\boldsymbol{x})$沿$\boldsymbol{e}^i$的方向导数。偏导数是方向导数的特例，它们的方向是各个坐标轴的正方向。由于$\nabla f(\boldsymbol{x})$与$\boldsymbol{e}^i$的内积是$\nabla f(\boldsymbol{x})$的第i分量$\nabla f(\boldsymbol{x})_i$，根据式（3.17）和式（3.19），有：

$$\nabla f(\boldsymbol{x})_i = \nabla f(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{e}^i = \nabla_{\boldsymbol{e}^i} f(\boldsymbol{x}) = \frac{\partial f}{\partial x_i}(\boldsymbol{x}), \quad i = 1, \cdots, n \quad (3.20)$$

所以梯度$\nabla f(\boldsymbol{x})$的第i分量是$f(\boldsymbol{x})$对自变量第i分量x_i的偏导数，于是梯度就是：

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \frac{\partial f}{\partial x_2}(\boldsymbol{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{pmatrix} \quad (3.21)$$

$f(\boldsymbol{x})$对自变量各个分量的偏导数是唯一的，所以$\nabla f(\boldsymbol{x})$也是唯一的。

3.1.4 驻点

函数$f(\boldsymbol{x})$的驻点（stationary point）是梯度为零向量的点。驻点也称临界点（critical point）。既然驻点的梯度是零向量，所以$f(\boldsymbol{x})$在驻点的切平面的法向量是：

$$\boldsymbol{w} = \begin{pmatrix} \nabla f(\boldsymbol{x}) \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix} \quad (3.22)$$

驻点的切平面的法向量$\boldsymbol{w}$垂直指向下方，所以切平面是水平的，如图 3-6 所示。$f(\boldsymbol{x})$在驻点沿任意方向$\boldsymbol{d}$的方向导数是$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}=0$，所以$f(\boldsymbol{x})$在驻点向任意方向的变化率都为 0。

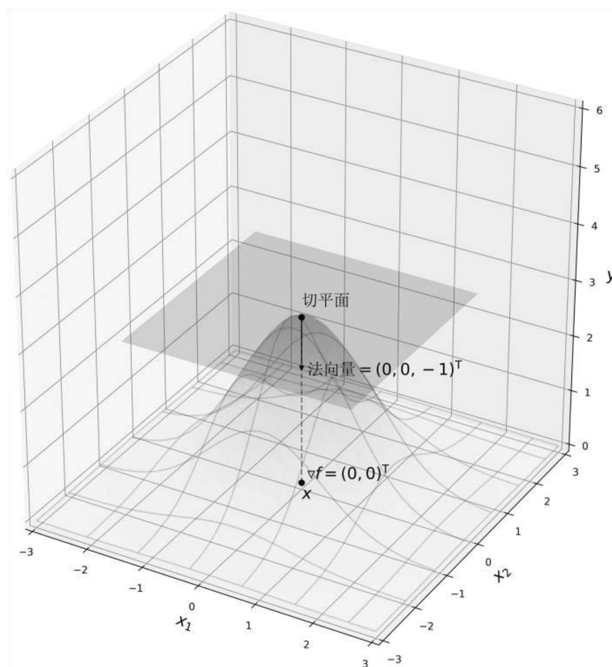


图 3-6　驻点的切平面

3.1.5　局部极小点

如果在点$\boldsymbol{x}^*$周围存在一个半径为$\varepsilon>0$的邻域，该邻域内所有点的函数值都不小于$f(\boldsymbol{x}^*)$，则$\boldsymbol{x}^*$是$f(\boldsymbol{x})$的一个局部极小点（local minima），用公式表示就是：

$$f(\boldsymbol{x})\geqslant f(\boldsymbol{x}^*),\quad \|\boldsymbol{x}-\boldsymbol{x}^*\|<\varepsilon \tag{3.23}$$

如果对于自变量空间的所有$\boldsymbol{x}$，都有$f(\boldsymbol{x})\geqslant f(\boldsymbol{x}^*)$，则$\boldsymbol{x}^*$是$f(\boldsymbol{x})$的全局最小点(global minima)。很显然，全局最小点是局部极小点。但是局部极小点不一定是全局最小点。类似还可以定义局部极大点（local maxima）和全局最大点（global maxima），如图 3-7 所示。

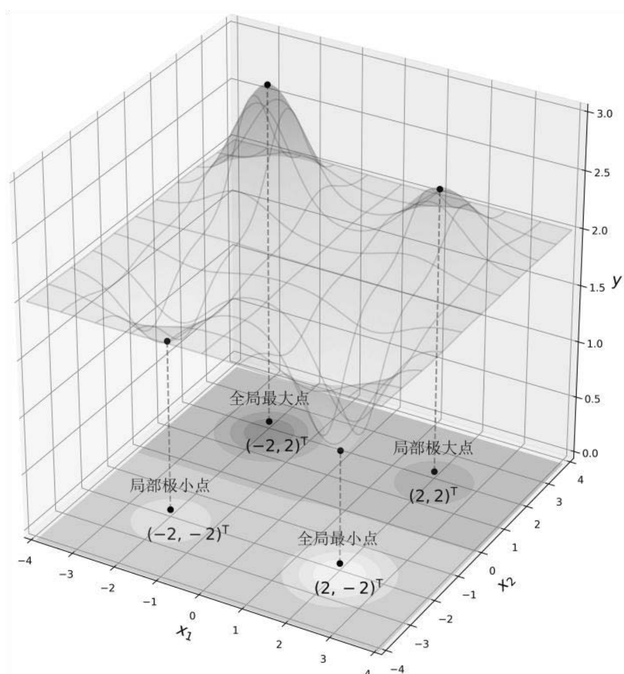


图 3-7　局部极小/大点和全局最小/大点

局部极小点$\boldsymbol{x}^*$一定是驻点，即$\|\nabla f(\boldsymbol{x}^*)\| = 0$。运用反证法，假设$\|\nabla f(\boldsymbol{x}^*)\| \neq 0$，从$\boldsymbol{x}^*$点出发沿$-\nabla f(\boldsymbol{x}^*)$方向做一个位移$-t\nabla f(\boldsymbol{x}^*)$，$t > 0$，因为$f(\boldsymbol{x})$在$\boldsymbol{x}^*$可导，有：

$$f\left(\boldsymbol{x}^* - t\nabla f(\boldsymbol{x}^*)\right) = f(\boldsymbol{x}^*) - t\|\nabla f(\boldsymbol{x}^*)\|^2 + \mathcal{R}\left(-t\nabla f(\boldsymbol{x}^*)\right) \tag{3.24}$$

$\mathcal{R}\left(-t\nabla f(\boldsymbol{x}^*)\right)$是位移$-t\nabla f(\boldsymbol{x}^*)$的高阶无穷小，于是有：

$$\lim_{t\to 0} \frac{-t\|\nabla f(\boldsymbol{x}^*)\|^2 + \mathcal{R}\left(-t\nabla f(\boldsymbol{x}^*)\right)}{\|-t\nabla f(\boldsymbol{x}^*)\|} = -\|\nabla f(\boldsymbol{x}^*)\| + \lim_{t\to 0} \frac{\mathcal{R}\left(-t\nabla f(\boldsymbol{x}^*)\right)}{\|-t\nabla f(\boldsymbol{x}^*)\|} = -\|\nabla f(\boldsymbol{x}^*)\| < 0 \tag{3.25}$$

这说明当t趋近于 0 时，式（3.24）的后两项的极限是负值。所以对于足够小的$\varepsilon > 0$，当$t < \varepsilon$时有：

$$f\left(\boldsymbol{x}^* - t\nabla f(\boldsymbol{x}^*)\right) - f(\boldsymbol{x}^*) = -t\|\nabla f(\boldsymbol{x}^*)\|^2 + \mathcal{R}\left(-t\nabla f(\boldsymbol{x}^*)\right) < 0 \tag{3.26}$$

随着t继续靠近 0，$\boldsymbol{x}^* - t\nabla f(\boldsymbol{x}^*)$在无限靠近$\boldsymbol{x}^*$的同时保持$f\left(\boldsymbol{x}^* - t\nabla f(\boldsymbol{x}^*)\right) < f(\boldsymbol{x}^*)$。这与$\boldsymbol{x}^*$是$f(\boldsymbol{x})$的局部极小点矛盾。所以$\nabla f(\boldsymbol{x}^*)$只能是零向量，即$\boldsymbol{x}^*$是驻点。类似可以证明，局部极大点也一定是驻点。驻点是局部极小点的必要非充分条件。驻点也有可能是局部极大点或者鞍点（saddle point）。鞍点的梯度也是零向量，但在任意一个邻域内都同时存在函数值更大的点和更小的点，

如图 3-8 所示。仅靠梯度难以判断驻点的类型，驻点的类型可由赫森矩阵的特征值揭示，这将在下一章讨论。

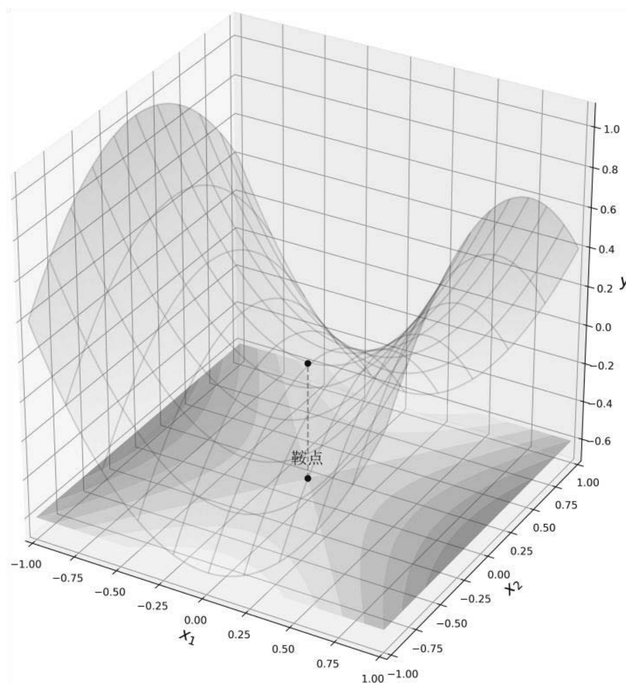


图 3-8　鞍点

3.2　梯度下降法

为了寻找函数的全局最小点，可以先找到满足必要条件的点——驻点。根据定义，可以求函数的梯度，令梯度为零向量而求得驻点，但大多数时候这并不可行。举个简单的例子，有一个二次型（quadratic form）函数：

$$f(x^1, x^2, \cdots, x^n) = \frac{1}{2}\sum_{i,j} w^{i,j} x^i x^j \tag{3.27}$$

其中，$w^{i,j} = w^{j,i}$。该二次型对每一个自变量x^i的偏导数是：

$$\frac{\partial f}{\partial x^i} = \sum_{j=1}^{n} w^{i,j} x^j \tag{3.28}$$

令梯度是零向量，求$x^1, x^2, \cdots, x^n$。这是解n元一次方程组：

$$\sum_{j=1}^{n} w^{i,j} x^j = 0\text{，}\quad i = 1, \cdots, n \tag{3.29}$$

一般情况下这需要$O(n^3)$的时间复杂度。当n非常大时（这在神经网络和深度学习中是必然的），求驻点的解析解是不可接受的。这还仅仅是简单的二次型的情况，当情况更复杂时，驻点的解析解有可能不存在，我们需要迭代的数值解法。

3.2.1 反梯度场

如果$f(\boldsymbol{x})$是n元函数，则$\nabla f(\boldsymbol{x})$是n维向量。可导函数$f(\boldsymbol{x})$在自变量空间中每一个点都有一个反梯度向量$-\nabla f(\boldsymbol{x})$，指向在该点函数值下降最快的方向。这就形成了一个速度（向量）场。可以将$-\nabla f(\boldsymbol{x})$画成箭头，尾部移到$\boldsymbol{x}$的位置，以这种方式将反梯度场呈现出来，如图 3-9 所示。

3

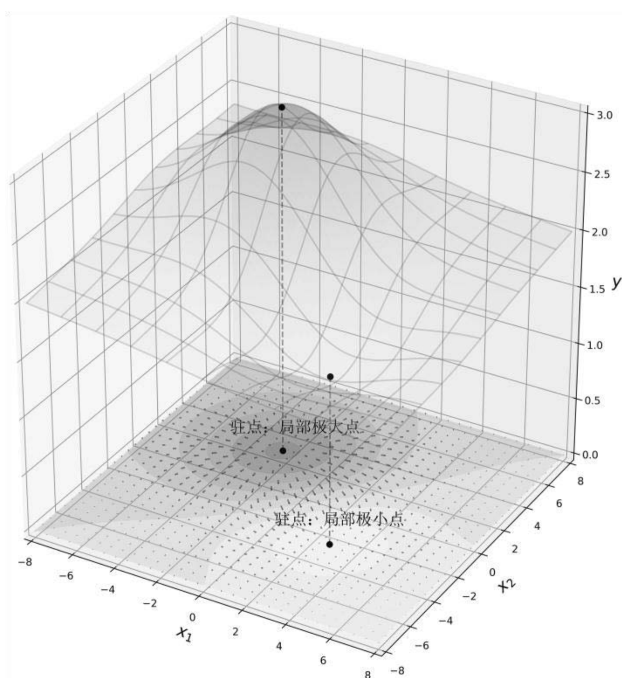


图 3-9 反梯度场

速度场在每一点指定了该位置的速度——方向和速率。在反梯度场的情况下，速度指向函数值下降最快的方向，速率大小是函数值的下降速率。将一个粒子（particle）从任意位置放入速度场中，它就会按照场指定的速度运动。在反梯度场的情况下，粒子就是按照梯度反方向，朝函数值下降最快的方向运动。

这里解释一下反梯度场的物理意义。在 2 维情况下可以将函数图像看作一幅起伏不平的地形，设重力加速度的大小是g，有一个质量是m的小球被放在任意位置。地面对小球的支持力垂直于

坡面，沿着该位置的切平面的朝上的法向量$(-\nabla f(\boldsymbol{x})_1, -\nabla f(\boldsymbol{x})_2, 1)^{\mathrm{T}}$。这个法向量与$x_1x_2$平面的夹角$\theta$的正切是$\tan\theta = \|\nabla f(\boldsymbol{x})\|^{-1}$。支持力的垂直分量抵消重力$m\mathrm{g}$，水平分量的大小则是$m\mathrm{g}/\tan\theta = m\mathrm{g}\|\nabla f(\boldsymbol{x})\|$，水平分量的方向是法向量向$x_1x_2$平面的投影$-\nabla f(\boldsymbol{x})$的方向。小球受到的水平作用力正是$-m\mathrm{g}\nabla f(\boldsymbol{x})$，小球的水平加速度就是$-\mathrm{g}\nabla f(\boldsymbol{x})$。所以若将函数图形看作地形，反梯度场是该地形的加速度场，而不是速度场，如图 3-10 所示。

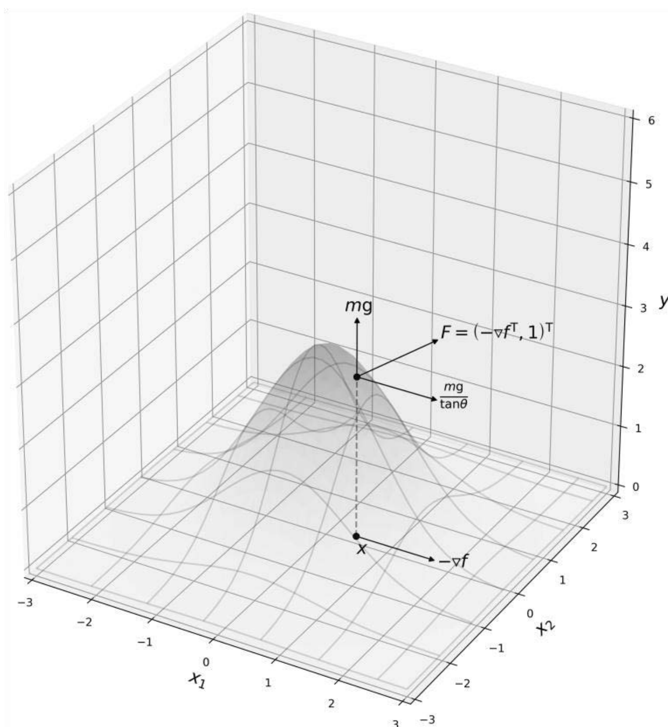


图 3-10　反梯度场的物理意义

局部极小点的梯度是零向量，它们是反梯度场的静止点。如果一个粒子处于静止点上，它将保持静止。同理，局部极大点也是静止点。局部极小点是稳定静止点，或者说吸引子（attractor）。当粒子偏离局部极小点一个小位移，它将被吸引回局部极小点。局部极大点是不稳定静止点，或者说排斥子（repeller）。当粒子位于局部极大点时，它是静止的，但是一旦有一个微小的扰动使它发生极小的位移，它将被推得远离局部极大点，如图 3-11 所示。

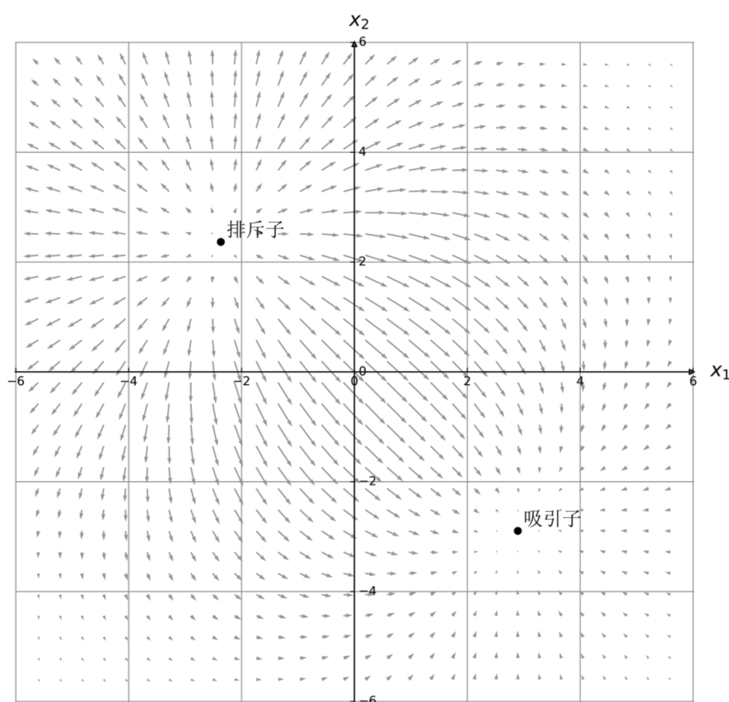


图 3-11　吸引子和排斥子

可以从任意位置开始模拟粒子在反梯度场中的运动。除非粒子的初始位置刚好是静止点，否则粒子将向函数值下降最快的方向，即反梯度方向运动，并最终逼近吸引子——局部极小点。从理论上，反梯度场只能保证粒子运动到局部极小点，不能保证运动到全局最小点，逼近的速度也没有保证。实际算法中无法精确模拟粒子的连续运动，而只能以离散迭代的方式近似，这会带来更多问题，甚至不收敛。

3.2.2　梯度下降法

在计算机中模拟粒子在速度场中的运动，这属于数值积分问题。梯度下降法（gradient descent，GD）是一种简单的数值积分算法，伪代码如下：

```
x^0 ← 随机初始化
t ← 0
while ‖∇f(x^t)‖ ⩾ ε:
    x^{t+1} ← x^t − η · ∇f(x^t)
    t ← t + 1
return x^t
```

$\varepsilon > 0$是一个预设的阈值，当$\|\nabla f(\boldsymbol{x})\| < \varepsilon$时，认为$\nabla f(\boldsymbol{x})$已经足够接近零向量，算法停止。也可以采用其他停止标准，例如循环次数达到预设的最大值，或者函数值的下降幅度小于阈值。$\eta > 0$是另一个预设值，称为学习率（learning rate，LR）或步长。每一次迭代中，自变量向$-\nabla f(\boldsymbol{x})$方向运动，运动的距离是$\eta \cdot \|\nabla f(\boldsymbol{x})\|$。

η是梯度下降法的一个超参数（hyper parameter）。我们可以保证在任意$\boldsymbol{x}$，能够找到一个合适的步长$\eta_{\boldsymbol{x}}$，使$f\big(\boldsymbol{x} - \eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})\big) < f(\boldsymbol{x})$，也就是说，保证从$\boldsymbol{x}$出发移动$-\eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})$，可以使函数值下降。这是因为：

$$f\big(\boldsymbol{x} - \eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})\big) = f(\boldsymbol{x}) - \eta_{\boldsymbol{x}} \|\nabla f(\boldsymbol{x})\|^2 + \mathcal{R}\big(-\eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})\big) \tag{3.30}$$

其中，$\mathcal{R}\big(-\eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})\big)$是$-\eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})$的高阶无穷小。式（3.30）与式（3.24）相似，可以证明存在一个$\varepsilon > 0$，当$\eta_{\boldsymbol{x}} < \varepsilon$时，满足$f\big(\boldsymbol{x} - \eta_{\boldsymbol{x}} \nabla f(\boldsymbol{x})\big) < f(\boldsymbol{x})$。我们说$-\nabla f(\boldsymbol{x})$是确保下降的方向，但确保下降只是理论上的，因为不同$\boldsymbol{x}$的$\eta_{\boldsymbol{x}}$不同，也无法计算出$\eta_{\boldsymbol{x}}$的值，所以只能使用固定步长η。在固定步长下，每一次更新不一定确保函数值下降。

3.2.3　梯度下降法的问题

因为梯度只包含函数的局部线性近似信息，在距离$\boldsymbol{x}$较远的地方，函数的形状会较大地偏离近似超平面，这是梯度所无法体现的。如果η设置得过大，函数值有可能不降反升。较小的η更有可能保证函数值下降，但是如果η过小，收敛的速度会很慢。函数的图像可能有千奇百怪的“地形”，很多病态情况都会对梯度下降法的效果产生负面影响，这里举几个例子。

“悬崖”（cliff）如图 3-12 所示。局部极小点位于悬崖脚下，靠近悬崖的位置具有较大梯度，这时梯度下降法会把点弹得远离悬崖，之后需要很多次迭代才能将点拉回悬崖脚下。

“峡谷”（valley）如图 3-13 所示。局部极小点在谷底，在峡谷壁上，反梯度方向并不指向局部极小点，而是指向峡谷对侧。这种情况下，梯度下降法会发生震荡，轻则延缓收敛速度，重则导致收敛失败。下一章介绍函数的局部二阶特性后，我们会知道峡谷的成因与赫森矩阵各个特征值的相对大小有关。

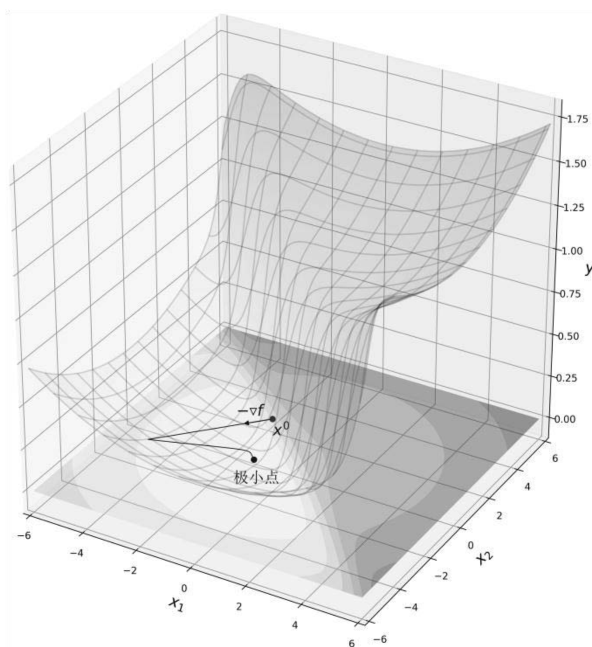


图 3-12 “悬崖”对梯度下降法的影响

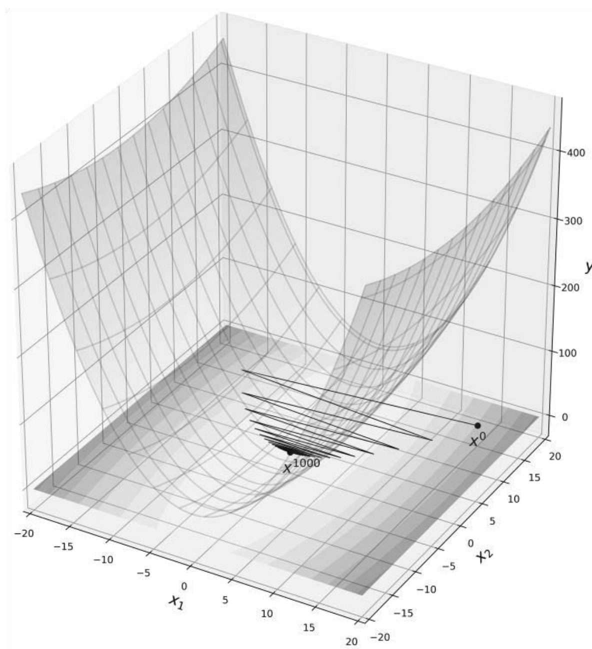


图 3-13 “峡谷”使梯度下降法发生震荡

“平原”（plain）如图 3-14 所示。在这样的区域里梯度的模非常小，这将导致收敛缓慢。本节仅举了几个简单直观的例子，来说明复杂函数地形下梯度下降法可能遇到的困难。实际的情况还要更复杂，也更难直接观察。

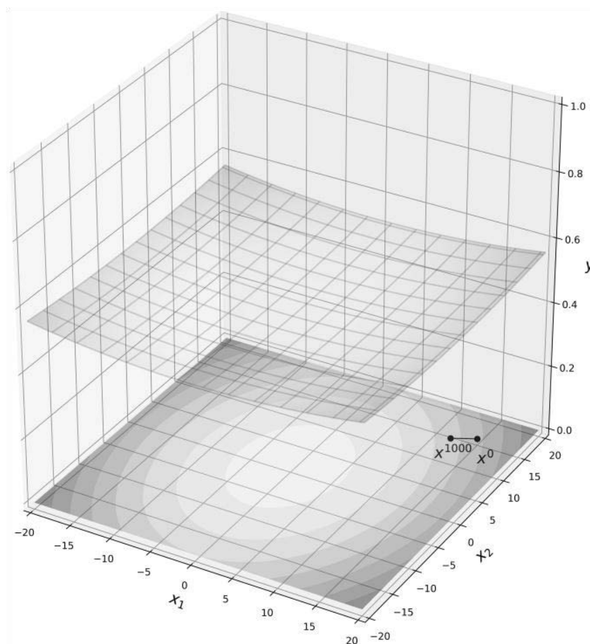


图 3-14　“平原”对梯度下降法的影响

3.3　梯度下降法的改进

针对梯度下降法的问题，人们提出了一些改进算法。这些算法有的着眼于调整学习率，有的对搜索方向做某种修正。它们都是原始梯度下降法的变体，终归还是利用局部一阶信息。有些变体会积累历史梯度信息，根据它们调整前进方向。积累梯度的历史也就是观察梯度的变化，尝试从梯度的历史变化中提取信息，模拟函数在当前位置的二阶信息。二阶优化算法用二次函数在局部模拟原函数，是一种更精确的近似。

梯度下降法的变体有很多，本节挑选了在原理上有代表性的，并有前后承接关系的五种进行讲解。它们是学习率调度、冲量法、AdaGrad、RMSProp 以及 Adam。

3.3.1　学习率调度

学习率是梯度下降法的一个重要超参数，它对算法的结果和效率都有重要的影响。原始梯度

下降法采用固定步长，如果步长设置得过大，则算法有可能不收敛，或者因震荡而延缓收敛。如果步长设置得过小，收敛则会非常缓慢，特别是在梯度较小的“平原”地带。学习率对算法的影响如图 3-15 所示。

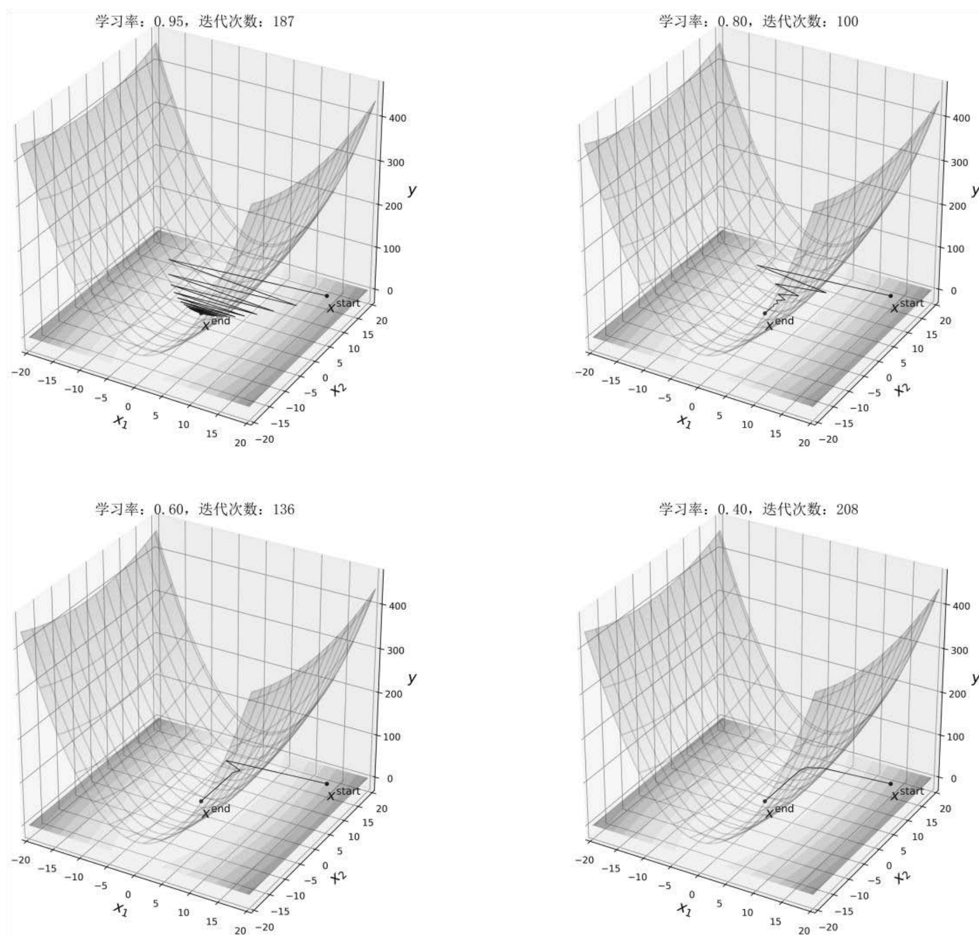


图 3-15　学习率对梯度下降法的影响

一个自然而然的想法就是抛弃固定的学习率，使学习率随着迭代而动态调整。例如，当发现最近若干轮迭代中函数值下降较快，则说明当前区域梯度较大，这时缩小步长；反之如果发现最近若干轮迭代中函数值下降很慢，则说明当前区域梯度较小，这时增加步长。或者当训练开始时采用较大的学习率，随着迭代逐渐缩小学习率。训练开始时可以假设当前距离目标尚远，采用较大的学习率可以加快收敛速度。而随着训练的进行，点逐渐向目标靠近，这时减小学习率以防止震荡。当然距离目标是远还是近在训练时是不知道的，所以这只是一种启发式的策略。这类办法

统称学习率调度（learning rate scheduling）。

实验表明，简单的学习率指数衰减策略（exponential decay scheduling）就能取得不错的效果。学习率指数衰减的公式为：

$$\eta^t \leftarrow \eta^{\text{init}} \cdot 10^{-\frac{t}{r}} \tag{3.31}$$

其中，η^{init}是初始学习率，t是迭代步数，超参数r控制衰减的速度。r越大，则衰减越慢。动态学习率η^t随迭代步数t变化的典型曲线如图 3-16 所示。

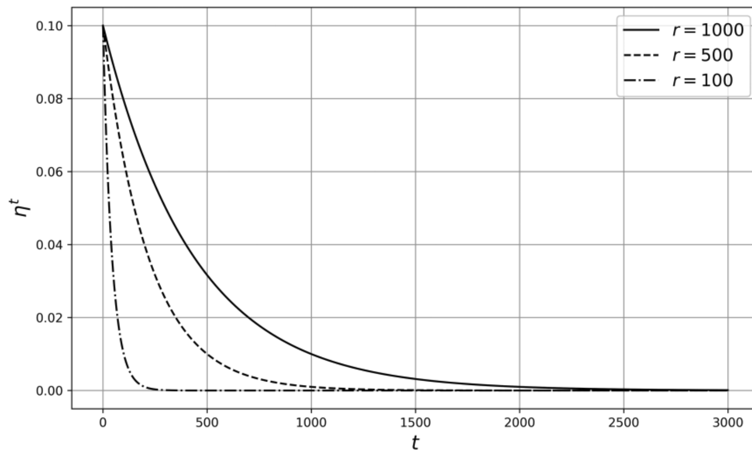


图 3-16 指数衰减的动态学习率随迭代步数变化的曲线

将学习率指数衰减更新式（3.31）插入到原始梯度下降算法中，伪代码如下：

```
x^0 ← 随机初始化
t ← 0
while ‖∇f(x^t)‖ ⩾ ε:
    η^t ← η^init · 10^(-t/r)
    x^(t+1) ← x^t − η^t · ∇f(x^t)
    t ← t + 1
return x^t
```

3.3.2 冲量法

3.2.1 节解释了反梯度场的物理意义，如果将函数图像视作重力场中的地形，那么反梯度场则是水平加速度场，而不是速度场。原始梯度下降法相当于将加速度场当作速度场，模拟粒子在其中的运动。冲量（momentum）法则恢复了反梯度场的加速度场本质，以反梯度向量为水平加

速度来模拟粒子的运动，伪代码如下：

```
x⁰ ← 随机初始化
v⁰ ← 0
t ← 0
while ‖∇f(xᵗ)‖ ⩾ ε:
    v^{t+1} ← βvᵗ − η·∇f(xᵗ)
    x^{t+1} ← xᵗ + v^{t+1}
    t ← t + 1
return xᵗ
```


从物理角度阐释，学习率η就是一个时间单元，反梯度向量$-\nabla f(\boldsymbol{x}^t)$现在是加速度向量，速度向量$\boldsymbol{v}^t$累积了从运动开始以来的速度变化。为了避免速度变得无限大，冲量法引入了摩擦。$\boldsymbol{v}^t$的更新式中的$0<\beta<1$相当于摩擦系数（实际上它是物理意义上的摩擦系数的一个变体）。如果梯度不变，是常向量∇f，则有：

$$\boldsymbol{v}^{\infty}=-(\eta\cdot\textstyle\sum_{i=0}^{\infty}\beta^i)\cdot\nabla f=-\frac{\eta\cdot\nabla f}{1-\beta} \tag{3.32}$$

可见如果梯度保持不变，冲量法的极限更新量是原始梯度下降法的$\frac{1}{1-\beta}$。如果$\beta=0.9$，则冲量法最终将趋近原始梯度下降法 10 倍的更新速度。如果梯度方向有变化，冲量法通过将之前的梯度进行滑动平均，能起到减少震荡的作用。

冲量法有一个变体：Nesterov 法。它与冲量法的区别在于$\boldsymbol{v}^{t+1}$更新式中的梯度不是在$\boldsymbol{x}^t$处计算，而是在$\boldsymbol{x}^t+\beta\boldsymbol{v}^t$处计算。因为冲量法的更新式可以写成$\boldsymbol{x}^{t+1}\leftarrow\boldsymbol{x}^t+\beta\boldsymbol{v}^t-\eta\nabla f$，相当于从$\boldsymbol{x}^t+\beta\boldsymbol{v}^t$出发前进$-\eta\nabla f$，所以使用$\boldsymbol{x}^t+\beta\boldsymbol{v}^t$处的梯度会更合适。

3.3.3　AdaGrad

原始梯度下降法对反梯度向量的每一个分量使用同样的学习率。用标量η乘$-\nabla f(\boldsymbol{x}^t)$，是对其长度进行缩放，但不改变其方向。AdaGrad 方法会为反梯度向量的每一个分量适配一个不同的学习率，这样就等于对前进方向做了调整，偏离了反梯度方向。先看伪代码：

```
x⁰ ← 随机初始化
s⁰ ← 0
t ← 0
while ‖∇f(xᵗ)‖ ⩾ ε:
    s^{t+1} ← sᵗ + ∇f(xᵗ) ⊗ ∇f(xᵗ)
    x^{t+1} ← xᵗ − η·∇f(xᵗ) ⊘ √(s^{t+1} ⊕ ϵ)
    t ← t + 1
return xᵗ
```

$\otimes$、$\oslash$和$\oplus$分别表示将两个向量的对应元素相乘、相除和相加。$\nabla f(\boldsymbol{x}^t)\otimes\nabla f(\boldsymbol{x}^t)$计算$f(\boldsymbol{x})$在$\boldsymbol{x}^t$的梯度的每个分量的平方，将其累加到向量$\boldsymbol{s}^{t+1}$中。$\boldsymbol{\epsilon}$是一个所有分量都是微小值（例如$10^{-18}$）的向量。更新自变量时，将$\boldsymbol{s}^{t+1}$与$\boldsymbol{\epsilon}$的对应分量相加后开方，除梯度的每一分量，得到前进方向。

$\boldsymbol{s}^{t+1}$的每个分量是算法执行至此时梯度每个分量的非中心方差（二阶矩）的累加。如果梯度的某一分量一直较大，则 AdaGrad 会对当前梯度的这个分量做压缩，减小该分量上的更新量，对前进方向进行调整，如图 3-17 所示。

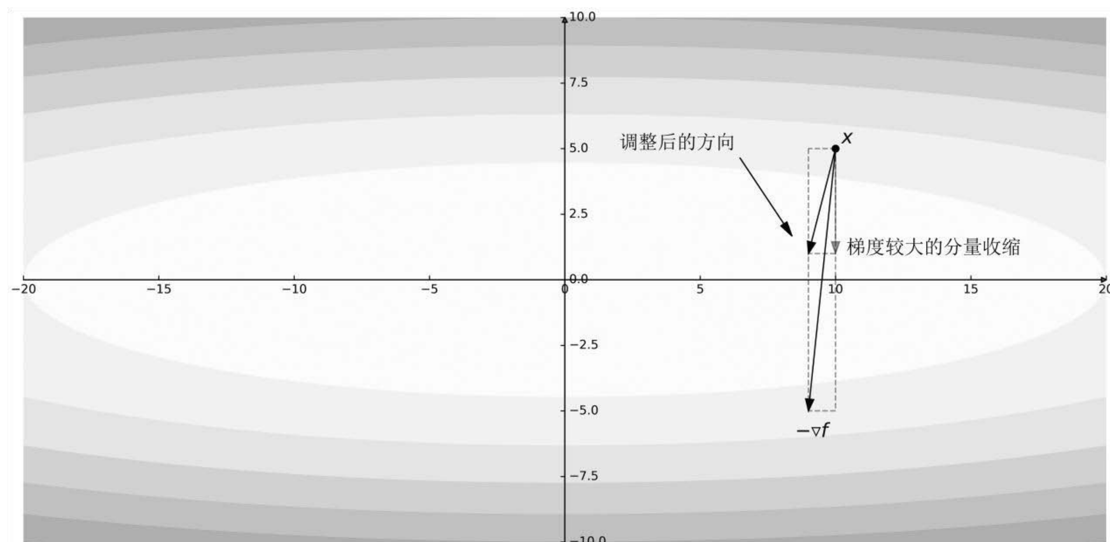


图 3-17　AdaGrad 对梯度一直较大的分量进行惩罚

下一章介绍牛顿法后，会看出 AdaGrad 是试图对函数局部二阶信息进行粗糙的估计。因为 AdaGrad 毕竟没有计算二阶信息，而只是用历史一阶信息模拟二阶信息，所以它还是属于一阶优化算法。因为累加项永远是正值，$\boldsymbol{s}^{t+1}$的各元素只增不减，所以 AdaGrad 还有学习率衰减的效果，但如果不加以限制，算法最终将陷入停滞。

3.3.4　RMSProp

AdaGrad 算法对梯度的每一个分量积累历史上所有值的平方和，调整前进方向时，各分量会根据自己的历史值平方和而获得不同的权值，历史值平方和较大的梯度分量会受到较大的惩罚。但使用梯度的全部历史对当前梯度进行修正是不适当的，应该使用近期一个时间窗内的梯度历史信息。时间上的局部导致空间上的局部，最近时间窗内的梯度历史反映的是函数的线性近似在局部的变化情况，这是对函数局部二阶信息更合理的模拟。

另外，无衰减地积累梯度分量的历史平方和，会导致权重趋近于 0，这起到类似学习率衰减的作用，但也会使算法最终陷入停滞。针对这些问题，RMSProp 对 AdaGrad 进行了改进，其伪代码如下：

```
x^0 ← 随机初始化
s^0 ← 0
t ← 0
while ‖∇f(x^t)‖ ⩾ ε:
    s^{t+1} ← βs^t + (1 − β) · ∇f(x^t) ⊗ ∇f(x^t)
    x^{t+1} ← x^t − η · ∇f(x^t) ⊘ √(s^{t+1} ⊕ ϵ)
    t ← t + 1
return x^t
```


RMSProp 引入了一个滑动平均系数$0 < \beta < 1$。每次累加梯度分量平方时，用β和$1 - \beta$为历史累加值和当前值加权。RMSProp 引入了一个新的超参数β，一般情况取$\beta = 0.9$即可。

3.3.5 Adam

Adam 是 adaptive moment estimation 的缩写。Adam 算法结合了冲量法和 RMSProp 的思想。先来看它的伪代码：

```
x^0 ← 随机初始化
s^0 ← 0
v^0 ← 0
t ← 0
while ‖∇f(x^t)‖ ⩾ ε:
    v^{t+1} ← β^1 v^t + (1 − β^1) · ∇f(x^t)
    s^{t+1} ← β^2 s^t + (1 − β^2) · ∇f(x^t) ⊗ ∇f(x^t)
    v^{t+1} ← v^{t+1} / (1−(β^1)^{t+1})
    s^{t+1} ← s^{t+1} / (1−(β^2)^{t+1})
    x^{t+1} ← x^t − η · v^{t+1} ⊘ √(s^{t+1} ⊕ ϵ)
    t ← t + 1
return x^t
```

在每一轮迭代中，Adam 以系数$0 < \beta^1 < 1$计算历史梯度的滑动平均$\boldsymbol{v}^{t+1}$，以系数$0 < \beta^2 < 1$计算梯度各分量平方的滑动平均$\boldsymbol{s}^{t+1}$。因为$\boldsymbol{v}^0$和$\boldsymbol{s}^0$用零向量初始化，且β^1和β^2接近 1，$\boldsymbol{v}^{t+1}$和$\boldsymbol{s}^{t+1}$在迭代初期会偏向零向量，故在迭代初期采用一个小于 1 的值做分母进行调整，使$\boldsymbol{v}^{t+1}$和$\boldsymbol{s}^{t+1}$远离零向量，随着迭代进行，分母趋近于 1，修正效果趋向于消失。Adam 用$\boldsymbol{v}^{t+1}$和$\boldsymbol{s}^{t+1}$更新自变量。

Adam 的超参数虽多，但默认值就可取得不错的效果，默认取$\beta^1 = 0.9$和$\beta^2 = 0.99$。由于$\boldsymbol{s}^t$的作用，迭代时的实际步长会发生衰减，所以η的取值不像在原始梯度下降中那样重要，一般

$\eta = 0.001$。图 3-18 比较了原始梯度下降法和几种变体的效果。注意在本示例的函数上，各种变体不见得优于原始梯度下降。

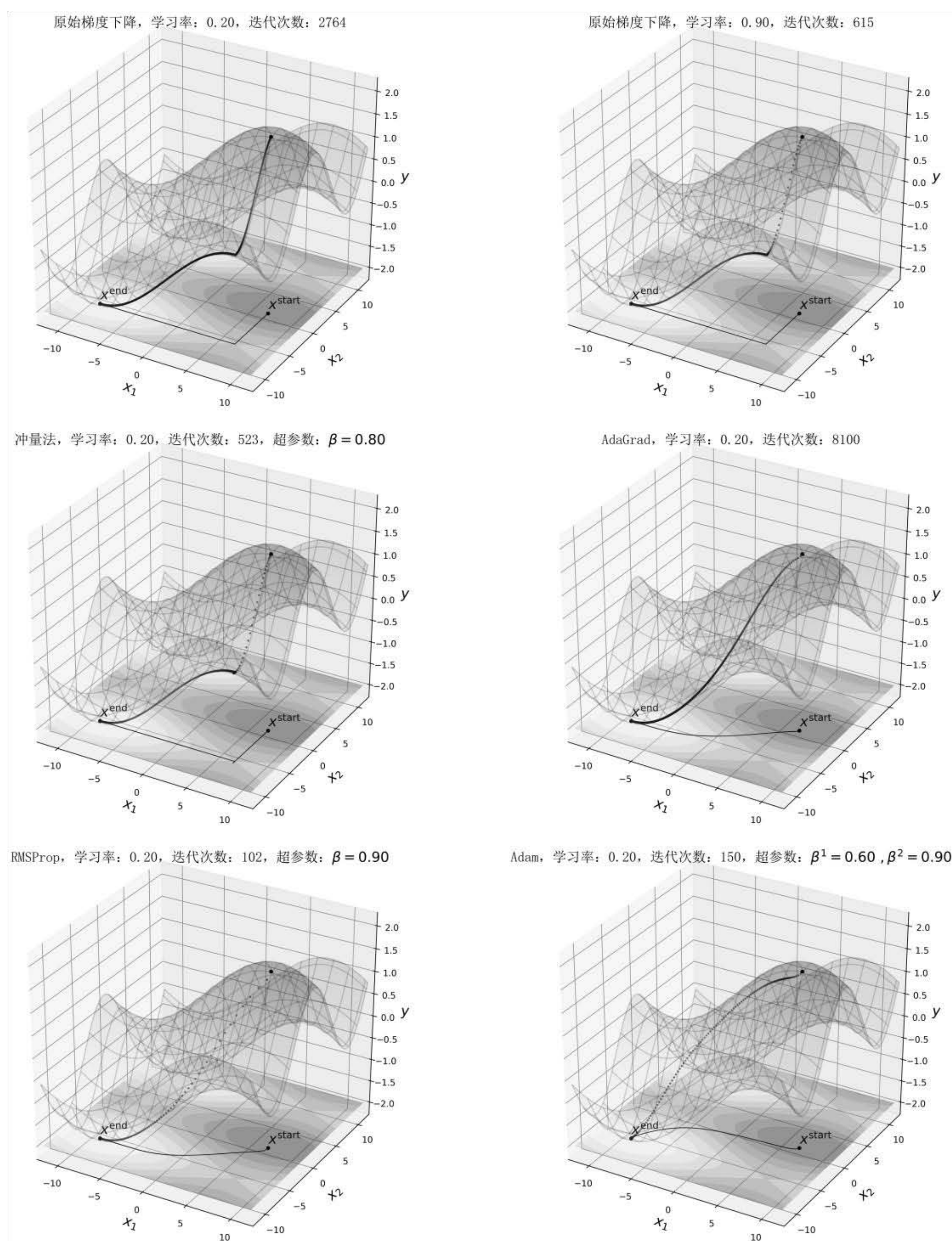


图 3-18　原始梯度下降法与几种变体

3.4 运用梯度下降法训练逻辑回归

运用梯度下降法训练逻辑回归，需要首先计算交叉熵损失对逻辑回归的参数w_i（$i=1,\cdots,n$）和b的梯度。计算梯度需要计算交叉熵损失对各个参数的偏导数。交叉熵损失是每一个训练样本的损失的平均，第i个训练样本$\{\boldsymbol{x}^i, y^i\}$的损失是：

$$\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i) = -y^i \log\frac{1}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}} - (1-y^i)\log\frac{1}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}} \quad (3.33)$$

3

现在计算$\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)$对$w_j$的偏导数。分两种情况考虑：训练样本属于正类，$y^i=1$；以及训练样本属于负类，$y^i=0$。当$y^i=1$时：

$$\frac{\partial\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)}{\partial w_j} = \frac{\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}{1+\mathrm{e}^{-b-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\left(-x_j^i\right) = \left(1-p^i(\boldsymbol{w}, b)\right)\left(-x_j^i\right) = -\left(y^i - p^i(\boldsymbol{w}, b)\right)x_j^i \quad (3.34)$$

其中，x_j^i是$\boldsymbol{x}^i$的第j分量，$p^i(\boldsymbol{w}, b)$是逻辑回归对$\boldsymbol{x}^i$的预测概率，将其视为关于$\boldsymbol{w}$和b的函数。当$y^i=0$时：

$$\frac{\partial\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)}{\partial w_j} = \frac{\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}{1+\mathrm{e}^{b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}x_j^i = p^i(\boldsymbol{w}, b)x_j^i = -\left(y^i - p^i(\boldsymbol{w}, b)\right)x_j^i \quad (3.35)$$

喜闻乐见的事情发生了，两种情况统一成一种情况：

$$\frac{\partial\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)}{\partial w_j} = -\left(y^i - p^i(\boldsymbol{w}, b)\right)x_j^i \quad (3.36)$$

在继续之前先观察一下式（3.36），括号中的$y^i - p^i(\boldsymbol{w}, b)$是真实标签（1 或 0）与预测概率之差。这个差可以看作模型在样本$\{\boldsymbol{x}^i, y^i\}$上的误差。更新参数时是加上负梯度，所以$\left(y^i - p^i(\boldsymbol{w}, b)\right)x_j^i$会被加到$w_j$上。$y^i - p^i(\boldsymbol{w}, b)$是真实值与预测概率的差距，这个差距以$x_j^i$为权重分配到$w_j$上。误差分配是看待梯度下降的一个视角，在后文介绍神经网络的反向传播算法时，会看到误差不仅在同一层内部分配，还要在层与层之间分配。

回到梯度计算中来，损失函数$\text{loss}(\boldsymbol{w}, b)$是对全部训练样本的损失做平均，所以$\text{loss}(\boldsymbol{w}, b)$对$w_j$的偏导数是每一个$\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)$对$w_j$的偏导数的平均：

$$\frac{\partial\text{loss}(\boldsymbol{w}, b)}{\partial w_j} = \frac{1}{M}\sum_{i=1}^{M}\frac{\partial\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)}{\partial w_j} = -\frac{1}{M}\sum_{i=1}^{M}\left(y^i - p^i(\boldsymbol{w}, b)\right)x_j^i \quad (3.37)$$

现在考察$\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)$对$b$的偏导数，类似的计算揭示$y^i=1$和$y^i=0$两种情况统一成一个表达式：

$$\frac{\partial\text{loss}(\boldsymbol{w}, b|\boldsymbol{x}^i, y^i)}{\partial b} = -\left(y^i - p^i(\boldsymbol{w}, b)\right) \quad (3.38)$$

于是损失函数$\text{loss}(\boldsymbol{w},b)$对$b$的偏导数就是：

$$\frac{\partial \text{loss}(\boldsymbol{w},b)}{\partial b}=\frac{1}{M}\sum_{i=1}^{M}\frac{\partial \text{loss}(\boldsymbol{w},b|\boldsymbol{x}^i,y^i)}{\partial b}=-\frac{1}{M}\sum_{i=1}^{M}\left(y^i-p^i(\boldsymbol{w},b)\right) \tag{3.39}$$

有了各个偏导数就可以计算$\text{loss}(\boldsymbol{w},b)$对$w_i$（$i=1,\cdots,n$）和$b$的梯度了：

$$\nabla \text{loss}(\boldsymbol{w},b)=-\frac{1}{M}\sum_{i=1}^{M}\left(y^i-p^i(\boldsymbol{w},b)\right)\begin{pmatrix}x_1^i\\x_2^i\\\vdots\\x_n^i\\1\end{pmatrix} \tag{3.40}$$

有了交叉熵损失对模型参数的梯度，就可以应用梯度下降法训练逻辑回归模型了，伪代码如下：

```
w^0 ← 随机初始化
b^0 ← 随机初始化
t ← 0
while ||∇loss(w^t, b^t)|| ≥ ε:
    w^{t+1} ← w^t + (η/M) Σ_{i=1}^{M} (y^i − p^i(w^t, b^t)) (x_1^i, x_2^i, ⋮, x_n^i)^T
    b^{t+1} ← b^t + (η/M) Σ_{i=1}^{M} (y^i − p^i(w^t, b^t))
    t ← t + 1
return w^t, b^t
```

上述训练过程本质上是根据交叉熵损失的梯度更新模型参数，但我们可以抛开损失函数和梯度概念来观察一下$\boldsymbol{w}$和b的更新公式。模型的输出$p^i(\boldsymbol{w},b)$是 0 和 1 之间的一个概率值，对于正类训练样本，$y^i=1$，我们希望提高$p^i(\boldsymbol{w},b)$，使它更接近 1。更新公式中$y^i-p^i(\boldsymbol{w},b)$大于 0，如果$x_j^i>0$，对$w_j$的更新是加上一个正值；如果$x_j^i<0$，对$w_j$的更新是加上一个负值。两种情况都是增加$w_j$与$x_j^i$之积，也就是增加$p^i(\boldsymbol{w},b)$。对于负类训练样本，$y^i=0$，我们希望降低$p^i(\boldsymbol{w},b)$，使它更接近 0。更新公式中$y^i-p^i(\boldsymbol{w},b)$小于 0，如果$x_j^i>0$，对$w_j$的更新是加上一个负值；如果$x_j^i<0$，对$w_j$的更新是加上一个正值。两种情况都是减小$w_j$与$x_j^i$之积，也就是减小$p^i(\boldsymbol{w},b)$。

对b的更新不依赖模型输入$\boldsymbol{x}$，正类训练样本增大b，负类训练样本压低b，变化的幅度与概率$p^i(\boldsymbol{w},b)$和理想值（1 或 0）的差距有关。训练样本上模型输出概率与理想值差距越大，则对b的影响越大。

每一个训练样本都将模型参数向对自己来说更理想的方向拉，梯度下降算法取所有训练样本的更新的平均，所有训练样本的“合力”将模型参数拉向能更好地分类训练集的方向。最早的感

知机模型的更新规则也是从类似的视角出发，但是每一次更新只取一个训练样本，将模型参数拉向它更理想的方向，之后再取下一个训练样本，如此循环往复。

3.5　梯度下降法训练逻辑回归的 Python 实现

本节，我们用原生 Python 以及 Numpy 库实现逻辑回归模型和梯度下降法及变体，将其用于鸟类生态类群二分类问题。注意，为了容易阅读并能清晰地展现原理，本书代码不做过多的抽象和封装。我们首先实现几个优化器，见如下代码：


```
import numpy as np


class Gradient:
    """
    原始梯度下降
    """
    def __init__(self, learning_rate = 0.001):
        self.learning_rate = learning_rate

    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量：学习率乘以梯度反方向
        """
        return -self.learning_rate * gradient


class Decay:
    """
    学习率衰减
    """
    def __init__(self, learning_rate = 0.001, r = 500):
        self.learning_rate = learning_rate
        self.r = r
        self.global_steps = 0


    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量。根据学习率衰减公式调整学习率
        """
        eta = self.learning_rate * 10 ** (-self.global_steps / self.r)
        self.global_steps += 1
        return -eta * gradient
```

```
class Momentum:
    """
    冲量梯度下降
    """
    def __init__(self, learning_rate = 0.001, beta = 0.9):
        self.learning_rate = learning_rate
        self.v = None
        self.beta = beta

    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量。加入冲量机制
        """
        if self.v is None:
            # 将 v 初始化为与梯度同维数的全零向量
            self.v = np.mat(np.zeros(gradient.shape[0])).T

        self.v = self.beta * self.v - self.learning_rate * gradient
        return self.v


class AdaGrad:
    """
    AdaGrad
    """
    def __init__(self, learning_rate = 0.001):
        self.learning_rate = learning_rate
        self.s = None

    def delta(self, gradient):
        """
        接受当前点的梯度，根据 AdaGrad 算法得到前进向量
        """
        if self.s is None:
            # 将 s 初始化为与梯度同维数的全零向量
            self.s = np.mat(np.zeros(gradient.shape[0])).T

        self.s = self.s + np.power(gradient, 2)
        return -self.learning_rate * gradient / np.sqrt(self.s + 1e-10)


class RMSProp:
    """
    RMSProp
    """
```

```
    def __init__(self, learning_rate = 0.001, beta = 0.9):
        self.learning_rate = learning_rate
        self.s = None
        self.beta = beta

    def delta(self, gradient):
        """
        接受当前点的梯度，根据 RMSProp 算法得到前进向量
        """
        if self.s is None:
            # 将 s 初始化为与梯度同维数的全零向量
            self.s = np.mat(np.zeros(gradient.shape[0])).T

        self.s = self.beta * self.s + (1 - self.beta) * np.power(gradient, 2)
        return -self.learning_rate * gradient / np.sqrt(self.s + 1e-10)


class Adam:
    """
    Adam
    """
    def __init__(self, learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.99):
        self.learning_rate = learning_rate
        self.s = None
        self.v = None
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.global_steps = 0

    def delta(self, gradient):
        """
        接受当前点的梯度，根据 Adam 算法得到前进向量
        """
        if self.s is None or self.v is None:
            # 将 s 初始化为与梯度同维数的全零向量
            self.s = np.mat(np.zeros(gradient.shape[0])).T
            # 将 v 初始化为与梯度同维数的全零向量
            self.v = np.mat(np.zeros(gradient.shape[0])).T

        self.v = self.beta_1 * self.v + (1 - self.beta_1) * gradient
        self.s = self.beta_2 * self.s + (1 - self.beta_2) * np.power(gradient, 2)

        self.v = self.v / (1 - self.beta_1 ** (self.global_steps + 1))
        self.s = self.s / (1 - self.beta_2 ** (self.global_steps + 1))
        self.global_steps += 1

        return -self.learning_rate * self.v / np.sqrt(self.s + 1e-10)
```

我们将原始梯度下降法、学习率衰减、冲量法、AdaGrad、RMSProp 和 Adam 封装成类，构造时传入相关超参数。这些类的 delta 方法接受当前梯度向量，返回更新向量。算法需要保存的状态都放在类的成员变量中。接下来我们实现逻辑回归，见如下代码：

```
from optimizer import *
import numpy as np


class LogisticRegression:

    def __init__(self, optimizer, iterations = 100000):

        assert(optimizer is not None)

        self.optimizer = optimizer
        self.iterations = iterations # 迭代次数

    def train(self, x, y):
        """
        x 为矩阵，形状是 n_samples * n_features ，每一行为一个样本
        y 为矩阵，形状是 n_samples * 1，元素为样本的标签，正类为 1， 负类为 0
        """

        # 在 x 最前面添加一列常数 1，作为偏置值的输入，以简化公式
        x = np.mat(np.c_[[1.0] * x.shape[0], x])

        # 根据 x 的列数（特征数）随机初始化权值，此时偏置值纳入了权值向量，相当于第一个权值
        # 权值向量为 n_features + 1 维向量，每个分量以 0 均值、0.01 标准差的正态分布初始化
        self.weights = np.mat(np.random.normal(0, 0.01, size=x.shape[1])).T

        for i in range(self.iterations):

            # 计算当前模型对训练集样本的输出
            p = self.predict(x, False)

            gradient = -x.T * (y - p) / x.shape[0] # 交叉熵损失对模型参数的梯度
            self.weights += self.optimizer.delta(gradient) # 更新模型参数

            # 评估当前模型并打印训练信息
            if i % 100 == 0:
                # 交叉熵损失
                cross_entropy = (-y.T * np.log(p) - (1.0 - y).T * np.log(1 - p)) / y.shape[0]

                # 正确率
                accuracy = np.sum(((p > 0.5).astype(np.int) == y).
```

```
                                astype(np.int)) / y.shape[0]

                print("迭代：{:d}，交叉熵：{:.6f}，正确率：{:.2f}%".format(
                    i + 1, cross_entropy[0, 0], accuracy * 100))


    def predict(self, x, augment = True):
        """
        预测函数。x 为矩阵，形状是 n_samples * n_features，每一行为一个样本
        augment 参数指示是否要在特征矩阵前添加一列常量 1
        """
        # 在 x 最前面添加一列常数 1，作为偏置值的输入
        if augment:
            x = np.mat(np.c_[[1.0] * x.shape[0], x])

        a = -np.matmul(x, self.weights)
        a[a > 1e2] = 1e2  # 防止数值过大
        p = 1.0 / (1.0 + np.power(np.e, a))

        # 剪裁概率值，保证其为合法的概率值
        p[p >= 1.0] = 1.0 - 1e-10
        p[p <= 0.0] = 1e-10

        return p
```

我们将逻辑回归算法封装成一个类 LogisticRegression，它的构造函数接受一个 optimizer 对象和迭代次数。每次迭代，计算训练集全体样本的平均交叉熵对模型参数的梯度，之后调用 optimizer 对象的 delta 方法，得到更新向量并更新参数。注意，我们给特征矩阵添加一列，列中所有元素都是常量 1。这相当于给每个样本添加一个永远为 1 的“特征”，将该“特征”作为偏置值的输入。这样做可以将偏置值纳入权值向量，简化公式。接下来我们将这个逻辑回归模型应用于鸟类生态类群问题，代码如下：

```
import pandas as pd
import numpy as np
from optimizer import *
from lr import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

# 读入数据
bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)

# 根据标签是否属于"SW", "W", "R"三类，构造二分类 1/0 标签
bird["type"] = bird.type.apply(lambda t: t in ["SW", "W", "R"]).astype(np.int)
```

```
data = bird.values
# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集
train_x = np.mat(data[:300,:-1])
train_y = np.mat(data[:300,-1]).T

# 其余样本作为测试集
test_x = np.mat(data[300:,:-1])
test_y = np.mat(data[300:,-1]).T

# 构造逻辑回归对象，优化器为 Adam，各超参数取默认值
lr = LogisticRegression(Adam())

# 在训练集上训练
lr.train(train_x, train_y)

# 对测试集进行预测
p = lr.predict(test_x) # 模型预测的正类概率
pred = (p > 0.5).astype(np.int) # 以 0.5 为阈值时，模型预测的类别

print("正确率：{:.2f}%，查准率：{:.2f}%，查全率：{:.2f}%，ROC 曲线下面积：{:.3f}".format(
        accuracy_score(test_y, pred) * 100,
        precision_score(test_y, pred) * 100,
        recall_score(test_y, pred) * 100,
        roc_auc_score(test_y.A, p)))
```

首先使用 pandas 库读入数据集 bird.csv 文件，去掉含有空值的行，再去掉名为“id”的列。之后，根据 type 列的值是否属于大型鸟而取 1 或 0，并重新赋值 type 列。将特征和标签当作 numpy 数组取出，随机扰乱行的顺序。取前 300 行作为训练集，后 113 行作为测试集。

构造 LogisticRegression 对象，传给构造函数一个 Adam 对象，超参数都取默认值，迭代数量也取默认值。调用 LogisticRegression 对象的 train 方法，参数是训练集的特征矩阵和标签向量。训练过程中，每 100 次迭代，打印当前模型在训练集上的交叉熵和正确率。若运行该代码，可以看到随着训练进行，训练集上的交叉熵在下降而正确率在上升。

训练完成后，用 LogisticRegression 对象对测试集样本做预测，输出预测为正类的概率。以 0.5 为概率阈值，得到模型对测试集预测的类别。之后，调用 scikit-learn 库的 accuracy_score、precision_score、recall_score 以及 roc_auc_score 函数计算模型在测试集上的正确率、查准率、

查全率以及 ROC 曲线下的面积。读者可以自行试验这份代码，替换不同的 `optimizer` 并观察模型的表现。

3.6 小结

梯度包含了多元函数的局部一阶近似信息。根据梯度可以得到函数沿自变量空间中任意方向的变化率。函数在某一点梯度决定了函数在该点的切平面的法向量。当自变量为 2 维时，我们可以直观地从函数图像中看出，梯度指向的方向是该点切平面的上坡方向。函数沿梯度方向具有最大的、正的方向导数。梯度的反方向是该点切平面的下坡方向，函数沿梯度反方向具有最小的、负的方向导数。梯度反方向是函数值下降最快的方向，在每一次迭代中，沿着当前点的梯度反方向运动一个距离，这就是梯度下降法。

仿射函数是一次函数，没有二次或高次项，其图像是“平直”的——直线、平面和超平面。线性近似是函数在局部的最粗糙的近似，当自变量远离当前点，基于当前点梯度的线性近似有可能与原函数大相径庭，所以梯度下降法是一种缺乏远见的贪心算法。如果用更高次的函数——例如二次函数——在局部拟合原函数，则可以得到对原函数的更精确的近似，以及关于函数局部形态的更丰富的信息，这称为二阶近似。下一章我们将介绍基于函数局部二阶信息的优化算法。

第 4 章

超越梯度下降

梯度下降法只参考函数的局部一阶特性，本章将介绍多元函数的局部二阶特性以及基于二阶特性的优化算法。想要理解函数的局部二阶特性，必须先掌握矩阵这个工具。本章首先回顾矩阵的相关知识，包括矩阵的定义、运算、逆矩阵、特征值与特征向量、谱分解、奇异值分解以及二次型。

函数的局部一阶特性蕴含在梯度向量之中，局部二阶特性蕴含在赫森矩阵之中。利用梯度可以在某点附近以仿射函数近似原函数，这个仿射函数是原函数在该点的一阶泰勒展开。更进一步，利用赫森矩阵可以在某点附近以二次函数近似原函数，这个二次函数是原函数在该点的二阶泰勒展开。只靠梯度无法确定驻点的类型，而如果赫森矩阵非奇异，则可以用赫森矩阵的特征值确定驻点的类型。总之，利用二阶特性可以得到更多关于函数局部形态的信息。

掌握了相关数学工具后，我们介绍两个基于函数局部二阶特性的优化算法：牛顿法和共轭方向法。我们将细致讲解这两种算法的原理和实现。最后，我们介绍如何运用牛顿法训练逻辑回归模型。通过本章，读者可以对函数的局部形态有更深刻的理解，并掌握二阶优化算法的精髓。

4.1　矩阵

本节首先回顾矩阵的相关知识。这不是一篇关于矩阵的全面介绍，而只包含理解赫森矩阵所必需的知识点。本节是 1.2 节的延续，从向量进入矩阵，阅读完本节，读者就具备了理解本书后续内容的全部线性代数知识。

4.1.1　矩阵基础

矩阵（matrix）是由标量构成的 2 维阵列。以一个 3×3 矩阵为例：

$$\boldsymbol{A}=\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}=(\boldsymbol{a}_{*,1} \quad \boldsymbol{a}_{*,2} \quad \boldsymbol{a}_{*,3})=\begin{pmatrix} \boldsymbol{a}_{1,*}^{\mathrm{T}} \\ \boldsymbol{a}_{2,*}^{\mathrm{T}} \\ \boldsymbol{a}_{3,*}^{\mathrm{T}} \end{pmatrix} \tag{4.1}$$

本书用黑斜体大写字母表示矩阵，例如$\boldsymbol{A}$。标量$a_{i,j}$是矩阵$\boldsymbol{A}$的第i行、第j列元素。$\boldsymbol{a}_{*,j}$表示矩阵$\boldsymbol{A}$的第j列，它是一个列向量：

$$\boldsymbol{a}_{*,j}=\begin{pmatrix} a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} \tag{4.2}$$

$\boldsymbol{a}_{i,*}$也是一个列向量，它的转置是矩阵$\boldsymbol{A}$的第i行：

$$\boldsymbol{a}_{i,*}=\begin{pmatrix} a_{i,1} \\ a_{i,2} \\ a_{i,3} \end{pmatrix} \tag{4.3}$$

矩阵的行本来是行向量，但我们用列向量$\boldsymbol{a}_{i,*}$表示这个行向量的转置。矩阵的行数和列数不一定相等，其形状可以是$m \times n$（$m \neq n$）：

$$\boldsymbol{A}_{m\times n}=\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \tag{4.4}$$

在不会引起混淆时，一般可省略脚标$m \times n$。两个相同形状的矩阵可以相加：

$$\boldsymbol{A}+\boldsymbol{B}=\begin{pmatrix} a_{1,1}+b_{1,1} & \cdots & a_{1,n}+b_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1}+b_{m,1} & \cdots & a_{m,n}+b_{m,n} \end{pmatrix} \tag{4.5}$$

矩阵相加就是把它们的对应元素相加。可以用标量k乘一个矩阵：

$$k\boldsymbol{A}=\begin{pmatrix} ka_{1,1} & \cdots & ka_{1,n} \\ \vdots & \ddots & \vdots \\ ka_{m,1} & \cdots & ka_{m,n} \end{pmatrix} \tag{4.6}$$

$-\boldsymbol{A}$就是用-1乘$\boldsymbol{A}$。显然有$\boldsymbol{A}-\boldsymbol{A}=\boldsymbol{A}+(-\boldsymbol{A})=\boldsymbol{O}$。$\boldsymbol{O}$是所有元素都为 0 的矩阵——零矩阵。矩阵$\boldsymbol{A}$的转置定义为：

$$\boldsymbol{A}^{\mathrm{T}}=\begin{pmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{pmatrix}=(\boldsymbol{a}_{1,*} \quad \boldsymbol{a}_{2,*} \quad \boldsymbol{a}_{3,*})=\begin{pmatrix} \boldsymbol{a}_{*,1}^{\mathrm{T}} \\ \boldsymbol{a}_{*,2}^{\mathrm{T}} \\ \boldsymbol{a}_{*,3}^{\mathrm{T}} \end{pmatrix} \tag{4.7}$$

$\boldsymbol{A}^{\mathrm{T}}$把$\boldsymbol{A}$的行当作列，把$\boldsymbol{A}$的列当作行。如果$\boldsymbol{A}$是$m \times n$的，那么$\boldsymbol{A}^{\mathrm{T}}$就是$n \times m$的。矩阵可以与

向量相乘。如果矩阵$\boldsymbol{A}$是$m \times n$的，则它可以与一个n维向量$\boldsymbol{x}$相乘：

$$\boldsymbol{Ax} = \sum_{j=1}^{n} x_j \, \boldsymbol{a}_{*,j} \tag{4.8}$$

矩阵$\boldsymbol{A}$与向量$\boldsymbol{x}$的乘积是用$\boldsymbol{x}$的分量对矩阵的列线性组合，所以$\boldsymbol{A}$的列数和$\boldsymbol{x}$的维数必须相同。$\boldsymbol{A}$与$\boldsymbol{x}$的乘积是一个m维向量。容易看出$\boldsymbol{Ax}$的第i个元素是$\sum_{j=1}^{n} x_j a_{i,j} = \boldsymbol{a}_{i,*}^{\mathrm{T}} \boldsymbol{x}$，即$\boldsymbol{A}$的第$i$行与$\boldsymbol{x}$的内积。有了矩阵和向量的乘积的定义，就可以定义矩阵与矩阵相乘：

$$\boldsymbol{AB} = (\boldsymbol{Ab}_{*,1} \quad \boldsymbol{Ab}_{*,2} \quad \cdots \quad \boldsymbol{Ab}_{*,k}) \tag{4.9}$$

$\boldsymbol{A}$与$\boldsymbol{B}$的乘积是矩阵，记作$\boldsymbol{AB}$。$\boldsymbol{AB}$的第j列是$\boldsymbol{A}$与$\boldsymbol{B}$的第j列的乘积。如果$\boldsymbol{A}$是$m \times n$的，那么$\boldsymbol{B}$的列$\boldsymbol{b}_{*,j}$（$j = 1, \cdots, k$）必须是n维向量，即$\boldsymbol{B}$必须为n行。只有这样$\boldsymbol{A}$才能与这些列相乘。而$\boldsymbol{B}$的列数k是任意的。所以要能够与$m \times n$的$\boldsymbol{A}$相乘，$\boldsymbol{B}$的形状必须是$n \times k$，k任意。容易看出，$\boldsymbol{AB}$的形状是$m \times k$。$\boldsymbol{AB}$的第i行、第j列元素是$\boldsymbol{A}$的第i行与$\boldsymbol{B}$的第j列的内积：

$$\boldsymbol{a}_{i,*}^{\mathrm{T}} \boldsymbol{b}_{*,j} = \sum_{s=1}^{n} a_{i,s} b_{s,j} \tag{4.10}$$

即使$\boldsymbol{A}$能与$\boldsymbol{B}$相乘，$\boldsymbol{B}$也不一定能与$\boldsymbol{A}$相乘，因为$\boldsymbol{B}$的列数k不一定等于$\boldsymbol{A}$的行数m。当$k = m$时，$\boldsymbol{B}$可以与$\boldsymbol{A}$相乘，但$\boldsymbol{BA}$不一定等于$\boldsymbol{AB}$，即矩阵乘法不满足交换律。一个反例就可以证明这一点，这里不再赘述。矩阵的乘法满足结合率：

$$(\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{A}(\boldsymbol{BC}) \tag{4.11}$$

矩阵乘法和加法满足分配律：

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}, \quad (\boldsymbol{A} + \boldsymbol{B})\boldsymbol{C} = \boldsymbol{AC} + \boldsymbol{BC} \tag{4.12}$$

矩阵乘法和标量乘法满足结合律：

$$\boldsymbol{A}(k\boldsymbol{B}) = (k\boldsymbol{A})\boldsymbol{B} = k(\boldsymbol{AB}) \tag{4.13}$$

矩阵的标量乘法对矩阵加法满足分配率：

$$k(\boldsymbol{A} + \boldsymbol{B}) = k\boldsymbol{A} + k\boldsymbol{B}, \quad (k + h)\boldsymbol{A} = k\boldsymbol{A} + h\boldsymbol{A} \tag{4.14}$$

矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的乘积的转置是：

$$(\boldsymbol{AB})^{\mathrm{T}} = \boldsymbol{B}^{\mathrm{T}} \boldsymbol{A}^{\mathrm{T}} \tag{4.15}$$

n维向量$\boldsymbol{x}$的转置$\boldsymbol{x}^{\mathrm{T}}$是行向量，也可视为$1 \times n$的矩阵，它可以乘另一个矩阵$\boldsymbol{A}$：

$$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} = (\boldsymbol{A}^{\mathrm{T}} \boldsymbol{x})^{\mathrm{T}} \tag{4.16}$$

上述几条定律的证明很简单，只需要检查一下等式两边每一个元素的值即可，此处省略细节。若把列向量和行向量分别看作$n \times 1$和$1 \times n$的矩阵，则矩阵与向量的乘法也满足这几条定律。行数和列数相同的矩阵称为方阵（square matrix），其形状为$n \times n$。方阵$\boldsymbol{A}$的对角线元素之和称为它的迹（trace）：

$$\operatorname{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{i,i} \tag{4.17}$$

对于方阵$\boldsymbol{A}$和$\boldsymbol{B}$，$\boldsymbol{AB}$的迹等于$\boldsymbol{BA}$的迹，这是因为：

$$\begin{aligned}\operatorname{tr}(\boldsymbol{AB}) &= \sum_{i=1}^{n} \boldsymbol{a}_{i,*}^{\mathrm{T}} \boldsymbol{b}_{*,i} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} b_{j,i} = \sum_{j=1}^{n} \sum_{i=1}^{n} b_{j,i} a_{i,j} \\ &= \sum_{j=1}^{n} \boldsymbol{b}_{j,*}^{\mathrm{T}} \boldsymbol{a}_{*,j} = \operatorname{tr}(\boldsymbol{BA})\end{aligned} \tag{4.18}$$

如果一个$n \times n$的方阵的对角线元素都是 1，其余元素都是 0，那么它称为单位矩阵（unit matrix），记作$\boldsymbol{I}_{n \times n}$：

$$\boldsymbol{I}_{n \times n} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} \tag{4.19}$$

容易验证对于任何矩阵$\boldsymbol{A}_{m \times n}$，都有$\boldsymbol{I}_{m \times m} \boldsymbol{A}_{m \times n} = \boldsymbol{A}_{m \times n} \boldsymbol{I}_{n \times n} = \boldsymbol{A}_{m \times n}$。在不会混淆时，一般省略$\boldsymbol{I}$的脚标。

4.1.2 矩阵的逆

如果对于$n \times n$方阵$\boldsymbol{A}$，存在一个同形状的方阵，记作$\boldsymbol{A}^{-1}$，满足：

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I} \tag{4.20}$$

则称$\boldsymbol{A}$是可逆的（invertible）或非奇异的（nonsingular）。$\boldsymbol{A}^{-1}$称为$\boldsymbol{A}$的逆矩阵。$\boldsymbol{A}$的逆矩阵是唯一的，因为假如矩阵$\boldsymbol{B}$也是$\boldsymbol{A}$的逆矩阵，根据定义有：

$$\boldsymbol{B} = \boldsymbol{IB} = \boldsymbol{A}^{-1}\boldsymbol{AB} = \boldsymbol{A}^{-1} \tag{4.21}$$

如果$\boldsymbol{A}$是可逆的，那么它的转置$\boldsymbol{A}^{\mathrm{T}}$也是可逆的，因为：

$$(\boldsymbol{A}^{-1})^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} = (\boldsymbol{A}\boldsymbol{A}^{-1})^{\mathrm{T}} = \boldsymbol{I},\quad \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}^{-1})^{\mathrm{T}} = (\boldsymbol{A}^{-1}\boldsymbol{A})^{\mathrm{T}} = \boldsymbol{I} \tag{4.22}$$

所以有$(\boldsymbol{A}^{\mathrm{T}})^{-1} = (\boldsymbol{A}^{-1})^{\mathrm{T}}$。可逆矩阵$\boldsymbol{A}$的列线性独立，因为假如$\boldsymbol{A}$的列$\boldsymbol{a}_{*,j}$（$j = 1, \cdots, n$）是线性相关的，则存在一组不全为 0 的系数$w^1, w^2, \cdots, w^n$，满足$\sum_{j=1}^{n} w^j \boldsymbol{a}_{*,j} = \boldsymbol{0}$。现在构造向量$\boldsymbol{w} = (w^1 \cdots w^n)^{\mathrm{T}}$。显然$\boldsymbol{w} \neq \boldsymbol{0}$，并且有：

$$\boldsymbol{A}\boldsymbol{w}=\mathbf{0} \tag{4.23}$$

因为$\boldsymbol{A}$是可逆的，所以必然存在$\boldsymbol{A}^{-1}$，满足：

$$\boldsymbol{w}=\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{w}=\boldsymbol{A}^{-1}\mathbf{0}=\mathbf{0} \tag{4.24}$$

这与$\boldsymbol{w}\neq\mathbf{0}$矛盾，所以可逆矩阵$\boldsymbol{A}$的列$\boldsymbol{a}_{*,j}$（$j=1,\cdots,n$）一定线性独立。如果方阵$\boldsymbol{A}$的逆矩阵是$\boldsymbol{A}^{\mathrm{T}}$，则称$\boldsymbol{A}$为正交矩阵：

$$\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}=\boldsymbol{I} \tag{4.25}$$

从式（4.25）可以看出正交矩阵$\boldsymbol{A}$的列$\boldsymbol{a}_{*,j}$（$j=1,\cdots,n$）是单位向量，且两两正交：

$$\begin{cases}\boldsymbol{a}_{*,i}^{\mathrm{T}}\boldsymbol{a}_{*,j}=0\text{，}\ i\neq j\\ \boldsymbol{a}_{*,i}^{\mathrm{T}}\boldsymbol{a}_{*,j}=1\text{，}\ i=j\end{cases} \tag{4.26}$$

因为正交矩阵$\boldsymbol{A}$是可逆的，所以它的列$\boldsymbol{a}_{*,j}$（$j=1,\cdots,n$）是线性独立的。这些列是n维线性空间$\mathbb{R}^n$的一组基，并且因为它们两两正交而且是单位向量，所以它们被称为$\mathbb{R}^n$的一组标准正交基。

如果一组向量$\boldsymbol{x}^i$（$i=1,\cdots,n$）是线性独立的，则可以通过施密特正交化过程（Schmidt orthogonalization）构造一组两两正交的向量$\widetilde{\boldsymbol{x}}^i$（$i=1,\cdots,n$）。具体过程是，首先令$\widetilde{\boldsymbol{x}}^1=\boldsymbol{x}^1$，然后令：

$$\widetilde{\boldsymbol{x}}^2=\boldsymbol{x}^2-\frac{(\boldsymbol{x}^2)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^1}{(\widetilde{\boldsymbol{x}}^1)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^1}\widetilde{\boldsymbol{x}}^1=\boldsymbol{x}^2-\frac{\|\boldsymbol{x}^2\|\cdot\cos\theta_{21}}{\|\widetilde{\boldsymbol{x}}^1\|}\widetilde{\boldsymbol{x}}^1 \tag{4.27}$$

其中，θ_{21}是$\boldsymbol{x}^2$与$\widetilde{\boldsymbol{x}}^1$的夹角。式（4.27）的本质是，$\widetilde{\boldsymbol{x}}^2$等于$\boldsymbol{x}^2$减去$\boldsymbol{x}^2$向$\widetilde{\boldsymbol{x}}^1$的投影。因为$\boldsymbol{x}^2$与$\widetilde{\boldsymbol{x}}^1=\boldsymbol{x}^1$线性独立，所以$\boldsymbol{x}^2$一定不是$\widetilde{\boldsymbol{x}}^1$的标量乘积，于是$\widetilde{\boldsymbol{x}}^2$不是零向量，而且容易验证$\widetilde{\boldsymbol{x}}^2$与$\widetilde{\boldsymbol{x}}^1$正交。接着再令：

$$\widetilde{\boldsymbol{x}}^3=\boldsymbol{x}^3-\frac{(\boldsymbol{x}^3)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^1}{(\widetilde{\boldsymbol{x}}^1)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^1}\widetilde{\boldsymbol{x}}^1-\frac{(\boldsymbol{x}^3)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^2}{(\widetilde{\boldsymbol{x}}^2)^{\mathrm{T}}\widetilde{\boldsymbol{x}}^2}\widetilde{\boldsymbol{x}}^2=\boldsymbol{x}^3-\frac{\|\boldsymbol{x}^3\|\cdot\cos\theta_{31}}{\|\widetilde{\boldsymbol{x}}^1\|}\widetilde{\boldsymbol{x}}^1-\frac{\|\boldsymbol{x}^3\|\cdot\cos\theta_{32}}{\|\widetilde{\boldsymbol{x}}^2\|}\widetilde{\boldsymbol{x}}^2 \tag{4.28}$$

其中，θ_{31}是$\boldsymbol{x}^3$与$\widetilde{\boldsymbol{x}}^1$的夹角，$\theta_{32}$是$\boldsymbol{x}^3$与$\widetilde{\boldsymbol{x}}^2$的夹角。式（4.28）的本质是，$\widetilde{\boldsymbol{x}}^3$等于$\boldsymbol{x}^3$减去$\boldsymbol{x}^3$向$\widetilde{\boldsymbol{x}}^1$和$\widetilde{\boldsymbol{x}}^2$张成空间的投影。$\widetilde{\boldsymbol{x}}^3$一定不是零向量，因为如果$\widetilde{\boldsymbol{x}}^3=\mathbf{0}$，那么$\boldsymbol{x}^3$就可以被$\widetilde{\boldsymbol{x}}^1$和$\widetilde{\boldsymbol{x}}^2$线性表出，也就可以被$\boldsymbol{x}^1$和$\boldsymbol{x}^2$线性表出，这与$\boldsymbol{x}^i$（$i=1,\cdots,n$）线性独立矛盾。容易验证$\widetilde{\boldsymbol{x}}^3$正交于$\widetilde{\boldsymbol{x}}^1$和$\widetilde{\boldsymbol{x}}^2$。

此过程继续下去，最终可构造一组正交向量$\widetilde{\boldsymbol{x}}^i$（$i=1,\cdots,n$）。因为它们两两正交，所以它们也线性独立。把$\widetilde{\boldsymbol{x}}^i$（$i=1,\cdots,n$）的每一个向量标准化（除以模），就得到了一组两两正交的单位向量，这就是施密特正交化过程。

4.1.3 特征值与特征向量

用方阵$\boldsymbol{A}$乘向量$\boldsymbol{x}$，相当于在$\mathbb{R}^n$中对$\boldsymbol{x}$做一个变换，将$\boldsymbol{x}$变换成$\boldsymbol{Ax}$，例如方阵：

$$\boldsymbol{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{4.29}$$

用$\boldsymbol{R}$乘向量$\boldsymbol{x}$等于将$\boldsymbol{x}$逆时针旋转θ度。任何方阵$\boldsymbol{A}$也改变不了零向量，因为$\boldsymbol{A0} = \boldsymbol{0}$。如果对于非零向量$\boldsymbol{v}$，$\boldsymbol{A}$只能改变$\boldsymbol{v}$的长度而不能改变它的方向，即存在某个标量$\lambda$（可以为 0），满足：

$$\boldsymbol{Av} = \lambda\boldsymbol{v} \tag{4.30}$$

则称λ是$\boldsymbol{A}$的特征值（eigenvalue），$\boldsymbol{v}$是$\boldsymbol{A}$的属于特征值λ的特征向量（eigenvector）。同一个特征向量不可能属于两个不同特征值，因为假如特征向量$\boldsymbol{v}$属于特征值λ^1和λ^2，且$\lambda^1 \neq \lambda^2$，那么：

$$(\lambda^1 - \lambda^2)\boldsymbol{v} = \lambda^1\boldsymbol{v} - \lambda^2\boldsymbol{v} = \boldsymbol{Av} - \boldsymbol{Av} = \boldsymbol{0} \tag{4.31}$$

但$\lambda^1 - \lambda^2 \neq 0$，则只可能$\boldsymbol{v} = \boldsymbol{0}$，这与特征向量是非零向量矛盾，所以同一个特征向量不可能属于两个不同的特征值,但同一个特征值可以拥有多个特征向量。如果$\boldsymbol{v}$是特征值λ的特征向量，容易验证$k\boldsymbol{v}$（$k \neq 0$）也是λ的特征向量。同一个特征值也可能拥有多个线性独立的特征向量。

令$\boldsymbol{v}$和$\boldsymbol{w}$是特征值λ的特征向量，如果$k\boldsymbol{v} + l\boldsymbol{w}$非 0，则它也是$\lambda$的特征向量，这很容易验证。如果特征值$\lambda$最多有$k$个线性独立的特征向量，那么由它们线性组合而得的非零向量也是λ的特征向量。这k个线性独立的特征向量张成的k维线性空间称为λ的特征空间，其中所有非零向量都是λ的特征向量。下面讲解如何求方阵$\boldsymbol{A}$的特征值和特征向量，将式（4.30）变形：

$$(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} = \boldsymbol{0} \tag{4.32}$$

如果λ是$\boldsymbol{A}$的特征值且$\boldsymbol{v}$是λ的特征向量，则它们满足式（4.32）。因为特征向量$\boldsymbol{v}$不是零向量，所以式（4.32）有非 0 解，即方阵$\boldsymbol{A} - \lambda\boldsymbol{I}$的列线性相关，这要求$\boldsymbol{A} - \lambda\boldsymbol{I}$的行列式$|\boldsymbol{A} - \lambda\boldsymbol{I}|$等于 0（本书不涉及行列式，关于这个结论可以查阅任何一种线性代数教材）。$|\boldsymbol{A} - \lambda\boldsymbol{I}| = 0$是关于$\lambda$的$n$次方程，它有$n$个根（包括重根和复数根），所以$\boldsymbol{A}$共有$n$个特征值（包括重复特征值和复数特征值）。得到$\lambda$后，解式（4.32）中的$\boldsymbol{v}$可以求得$\lambda$的特征向量。

$\boldsymbol{A}$的属于不同特征值的特征向量是线性独立的，我们来证明这个结论。令$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^k$分别是矩阵$\boldsymbol{A}$的$k$个不同特征值$\lambda^1, \lambda^2, \cdots, \lambda^k$的特征向量，若它们线性相关，则其中某一个$\boldsymbol{v}^s$可以被其他$\boldsymbol{v}^i$（$i \neq s$）线性表出：

$$\boldsymbol{v}^s = \textstyle\sum_{i \neq s} w^i \boldsymbol{v}^i \tag{4.33}$$

因为$\boldsymbol{v}^s$是特征向量，所以它不是零向量，于是w^i（$i \neq s$）一定不全为 0。根据特征值和特征向量的定义：

$$\lambda^s \boldsymbol{v}^s = \boldsymbol{A}\boldsymbol{v}^s = \sum_{i \neq s} w^i \boldsymbol{A}\boldsymbol{v}^i = \sum_{i \neq s} w^i \lambda^i \boldsymbol{v}^i \tag{4.34}$$

分两种情况讨论，第一种情况，如果$\lambda^s = 0$，那么其他$\lambda^i \neq 0$（$i \neq s$），另外有w^i（$i \neq s$）不全为 0，说明$\boldsymbol{v}^i$（$i \neq s$）是线性相关的。这样，在$\lambda^s = 0$的情况下，我们将问题规模缩小了 1。第二种情况，如果$\lambda^s \neq 0$，有：

$$\boldsymbol{v}^s = \sum_{i \neq s} w^i \frac{\lambda^i}{\lambda^s} \boldsymbol{v}^i \tag{4.35}$$

根据式（4.33）有：

$$\sum_{i \neq s} \left(w^i \frac{\lambda^i}{\lambda^s} \boldsymbol{v}^i - w^i \boldsymbol{v}^i \right) = \sum_{i \neq s} w^i \left(\frac{\lambda^i}{\lambda^s} - 1 \right) \boldsymbol{v}^i = \boldsymbol{0} \tag{4.36}$$

因为是不同的特征值，所以$\frac{\lambda^i}{\lambda^s} - 1 \neq 0$（$i \neq s$），另外有$w^i$（$i \neq s$）不全为 0，说明$\boldsymbol{v}^i$（$i \neq s$）是线性相关的。这样，在$\lambda^s \neq 0$的情况下我们也将问题规模减小了 1。

将这个过程持续下去，最终将只剩下两个向量$\boldsymbol{v}^i$和$\boldsymbol{v}^j$。它们分属不同的特征值λ^i和λ^j，且$\boldsymbol{v}^i$和$\boldsymbol{v}^j$线性相关。不妨假设$\boldsymbol{v}^i = k\boldsymbol{v}^j$，则$\boldsymbol{v}^i$也是$\lambda^j$的特征向量。之前已经证明，一个特征向量不可能同时属于两个不同的特征值，这就推翻了最早的假设，证明了分属不同特征值的特征向量是线性独立的。

4.1.4　对称矩阵的谱分解

如果方阵$\boldsymbol{A}$满足：

$$\boldsymbol{A} = \boldsymbol{A}^{\mathrm{T}} \tag{4.37}$$

则称$\boldsymbol{A}$是对称矩阵。如果对称矩阵$\boldsymbol{A}$的元素都是实数，则称它为实对称矩阵。实对称矩阵的特征值都是实数。为证明这个结论，我们需要暂时离开实数域。我们首先回忆一下复数。复数$\lambda = a + b\mathrm{i}$的共轭（conjugate）是$\bar{\lambda} = a - b\mathrm{i}$，也就是将虚部取反。$\lambda$与它的共轭$\bar{\lambda}$满足：

$$\lambda\bar{\lambda} = (a + b\mathrm{i})(a - b\mathrm{i}) = a^2 + b^2 \geqslant 0 \tag{4.38}$$

只有当λ的实部a和虚部b都为 0 时，才有$\lambda\bar{\lambda} = 0$，否则$\lambda\bar{\lambda} > 0$。对于两个复数$\lambda = a + b\mathrm{i}$和$\xi = c + d\mathrm{i}$，有：

$$\bar{\lambda}\bar{\xi} = (ac - bd) - (bc + ad)\mathrm{i} = \overline{\lambda\xi} \tag{4.39}$$

现在我们暂时允许矩阵的元素、特征值以及特征向量的分量是复数。把复矩阵$\boldsymbol{A}$的元素全都取共轭就得到$\boldsymbol{A}$的共轭$\bar{\boldsymbol{A}}$。如果复数λ和复向量$\boldsymbol{v}$是$\boldsymbol{A}$的特征值及相应特征向量，由式（4.39）容易得出，$\bar{\lambda}$和$\bar{\boldsymbol{v}}$是$\bar{\boldsymbol{A}}$的特征值及相应特征向量：

$$\bar{\boldsymbol{A}}\bar{\boldsymbol{v}} = \overline{\boldsymbol{A}\boldsymbol{v}} = \overline{\lambda\boldsymbol{v}} = \bar{\lambda}\bar{\boldsymbol{v}} \tag{4.40}$$

如果$\boldsymbol{A}$是实对称矩阵，有$\boldsymbol{A} = \bar{\boldsymbol{A}} = \boldsymbol{A}^{\mathrm{T}} = \bar{\boldsymbol{A}}^{\mathrm{T}}$，则有：

$$\lambda\bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{v} = \bar{\boldsymbol{v}}^{\mathrm{T}}(\lambda\boldsymbol{v}) = \bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{v} = \bar{\boldsymbol{v}}^{\mathrm{T}}\bar{\boldsymbol{A}}^{\mathrm{T}}\boldsymbol{v} = (\bar{\boldsymbol{A}}\bar{\boldsymbol{v}})^{\mathrm{T}}\boldsymbol{v} = \left(\bar{\lambda}\bar{\boldsymbol{v}}\right)^{\mathrm{T}}\boldsymbol{v} = \bar{\lambda}\bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{v} \tag{4.41}$$

因为$\boldsymbol{v} \neq \boldsymbol{0}$，根据式（4.38）有$\bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{v} > 0$，所以一定有$\lambda = \bar{\lambda}$，$\lambda$的虚部只能是 0，即$\lambda$是实数。$n$次方程$|\boldsymbol{A} - \lambda\boldsymbol{I}| = 0$在复数域内一定有$n$个根，允许重根。刚刚证明了实对称矩阵$\boldsymbol{A}$的特征值只能是实数，所以它有$n$个可重复的实特征值。现在我们可以离开复数域，从现在开始只讨论实数。有一个结论本书不加证明：如果λ是对称矩阵$\boldsymbol{A}$的k重特征值（即方程$|\boldsymbol{A} - \lambda\boldsymbol{I}| = 0$的$k$重根），则$\lambda$的特征空间的维度是$k$，即$\lambda$有$k$个线性独立的特征向量。

现在我们介绍对称矩阵的谱分解（spectral decomposition），又叫特征值分解（eigen decomposition）。首先得到对称矩阵$\boldsymbol{A}$的n个特征值$\lambda^1, \lambda^2, \cdots, \lambda^n$，并从大到小排列。为每个特征值找到一个单位特征向量。如果某个特征值是k重的，那么找到它的k个线性独立的特征向量，经过施密特正交化过程得到k个正交的单位向量。它们由原来的k个特征向量线性组合而得，所以仍然是该特征值的特征向量。这样一共可找到n个单位特征向量$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^n$。

对称矩阵$\boldsymbol{A}$的属于不同特征值的特征向量是正交的。因为，如果λ^i和λ^j是$\boldsymbol{A}$的两个不同特征值，$\boldsymbol{v}^i$和$\boldsymbol{v}^j$是它们各自的特征向量，有：

$$\lambda^j(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j = (\boldsymbol{v}^i)^{\mathrm{T}}(\lambda^j\boldsymbol{v}^j) = (\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{A}\boldsymbol{v}^j = (\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v}^j = (\boldsymbol{A}\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j = \lambda^i(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j \tag{4.42}$$

式（4.42）多次利用了特征值和特征向量的定义，以及$\boldsymbol{A}$是对称矩阵的事实。根据式（4.42）有：

$$(\lambda^i - \lambda^j)(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j = 0 \tag{4.43}$$

因为$\lambda^i - \lambda^j \neq 0$，所以必有$(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j = 0$，即$\boldsymbol{v}^i$和$\boldsymbol{v}^j$正交。在之前的构造过程中，$\boldsymbol{A}$的属于同一个特征值的多个特征向量已经过施密特正交化，它们彼此正交，刚刚又证明了$\boldsymbol{A}$的不同特征值的特征向量是正交的，所以单位特征向量$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^n$两两正交。因为它们两两正交，所以它们线性独立。用它们构造矩阵$\boldsymbol{V}^{\mathrm{T}}$：

$$\boldsymbol{V}^{\mathrm{T}} = (\boldsymbol{v}^1 \quad \cdots \quad \boldsymbol{v}^n) \tag{4.44}$$

因为$\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^n$是两两正交的单位向量，所以$\boldsymbol{V}^{\mathrm{T}}$是正交矩阵。构造对角矩阵：

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda^n \end{pmatrix} \tag{4.45}$$

$\boldsymbol{\Lambda}$的n个对角线元素是$\boldsymbol{A}$的从大到小排列的特征值，其余元素是 0，有：

$$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda} = (\boldsymbol{v}^1 \quad \cdots \quad \boldsymbol{v}^n)\begin{pmatrix} \lambda^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda^n \end{pmatrix} = (\lambda^1\boldsymbol{v}^1 \quad \cdots \quad \lambda^n\boldsymbol{v}^n) = \boldsymbol{A}\boldsymbol{V}^{\mathrm{T}} \tag{4.46}$$

因为$\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{I}$，所以有：

$$\boldsymbol{A} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V} \tag{4.47}$$

这就是对称矩阵$\boldsymbol{A}$的特征值分解，又称谱分解。

4.1.5　奇异值分解

令$\boldsymbol{X}$是$m \times n$矩阵，则$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$是$n \times n$的对称矩阵。$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$有$n$个（可重）非负特征值，将它们从大到小排列$\lambda^1 \geqslant \lambda^2 \geqslant \cdots \geqslant \lambda^n$。如果$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$有$r \leqslant n$个特征值大于 0，将它们记为$\lambda^1, \lambda^2, \cdots, \lambda^r$，它们的平方根$\sqrt{\lambda^1}, \sqrt{\lambda^2}, \cdots, \sqrt{\lambda^r}$称为$\boldsymbol{X}$的奇异值。$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$可以谱分解为$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}$。$\boldsymbol{V}^{\mathrm{T}}$的列是$\mathbb{R}^n$的一组标准正交基。构造$r$个向量$\boldsymbol{u}^1, \boldsymbol{u}^2, \cdots, \boldsymbol{u}^r$：

$$\boldsymbol{u}^i = \frac{\boldsymbol{X}\boldsymbol{v}_i}{\sqrt{\lambda^i}}, \quad i = 1, \cdots, r \tag{4.48}$$

其中，$\boldsymbol{u}^i$（$i = 1, \cdots, r$）是两两正交的标准向量，因为：

$$\langle \boldsymbol{u}^i, \boldsymbol{u}^j \rangle = \frac{\boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{v}_j}{\sqrt{\lambda^i\lambda^j}} = \frac{\boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{v}_j}{\sqrt{\lambda^i\lambda^j}} = \frac{\boldsymbol{e}_i{}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{e}_j}{\sqrt{\lambda^i\lambda^j}} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{4.49}$$

构造$m \times r$矩阵$\boldsymbol{U} = (\boldsymbol{u}^1 \quad \cdots \quad \boldsymbol{u}^r)$以及$r \times n$矩阵$\boldsymbol{\Gamma}$：

$$\boldsymbol{\Gamma} = \begin{pmatrix} \sqrt{\lambda^1} & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda^r} & 0 & \cdots & 0 \end{pmatrix} \tag{4.50}$$

因为$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$只有$r$个特征值非 0，对于后$n-r$个 0 特征值的特征向量，有$\boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{v}_i = \boldsymbol{v}_i{}^{\mathrm{T}}(0\boldsymbol{v}_i) = 0$（$i > r$）。$\boldsymbol{X}\boldsymbol{v}_i$与自身的内积为 0，所以$\boldsymbol{X}\boldsymbol{v}_i$是零向量，于是：

$$\boldsymbol{X}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{X}(\boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{v}_n) = (\boldsymbol{X}\boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{X}\boldsymbol{v}_r \quad 0 \quad \cdots \quad 0) = \boldsymbol{U}\boldsymbol{\Gamma} \tag{4.51}$$

因为$\boldsymbol{V}^{\mathrm{T}}$是正交矩阵，再根据式（4.51），有$\boldsymbol{X}=\boldsymbol{U\Gamma V}$，这就是$\boldsymbol{X}$的奇异值分解（singular value decomposition，SVD）。全体由$\boldsymbol{X}$的列线性组合而成的向量构成一个线性空间，称为$\boldsymbol{X}$的列空间。可以证明$\boldsymbol{X}$的列空间中的所有向量都可由$\boldsymbol{u}^i$($i=1,\cdots,r$)线性表出，因为对于任何一个n维向量$\boldsymbol{\beta}$，有：

$$\boldsymbol{X\beta}=\boldsymbol{U\Gamma V\beta} \tag{4.52}$$

$\boldsymbol{U}$的r个列$\boldsymbol{u}^i$（$i=1,\cdots,r$）是两两正交（于是线性独立）的标准向量，能线性表出$\boldsymbol{X}$的列空间的所有向量，所以它们是$\boldsymbol{X}$的列空间的一组标准正交基。

4.1.6 二次型

用某个方阵$\boldsymbol{A}$（不一定对称）构造多元函数$q(\boldsymbol{x})$：

$$q(\boldsymbol{x})=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Ax} \tag{4.53}$$

这种形式的函数称作二次型（quadratic form），例如：

$$q(\boldsymbol{x})=(x_1 \quad x_2)\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}={x_1}^2+5x_1x_2+4{x_2}^2 \tag{4.54}$$

二次型只包含二次项，不包含一次项、常数项（0次）以及更高次项。这种情况称二次型是“齐次”的。从式（4.54）可以看出，$q(\boldsymbol{x})$也可以写成：

$$q(\boldsymbol{x})=(x_1 \quad x_2)\begin{pmatrix}1 & \frac{5}{2}\\ \frac{5}{2} & 4\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}={x_1}^2+5x_1x_2+4{x_2}^2 \tag{4.55}$$

将“交叉”项（二元情况下就是x_1x_2）的系数对半分，则任何一个二次型都能写成关于对称矩阵的二次型。一个对称矩阵$\boldsymbol{A}$，如果对于任意向量$\boldsymbol{x}$都有：

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Ax}\geqslant 0 \tag{4.56}$$

则称$\boldsymbol{A}$是半正定的（positive semidefinite）。如果对任意$\boldsymbol{x}\neq\boldsymbol{0}$，不等号是严格的（$>$），则称$\boldsymbol{A}$是正定的（positive definite）。相应地还有半负定（negative semidefinite）和负定（negative definite）。半正定矩阵的所有特征值都不为负，因为假如$\boldsymbol{A}$有一个负特征值$\lambda^i<0$，$\boldsymbol{A}$可以分解为：

$$\boldsymbol{A}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda V} \tag{4.57}$$

因为$\boldsymbol{V}^{\mathrm{T}}$和$\boldsymbol{V}$是正交矩阵，所以$\boldsymbol{V}$的列是$\mathbb{R}^n$的一组标准正交基，存在$\boldsymbol{x}$使$\boldsymbol{e}^i=\boldsymbol{Vx}$，那么有：

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Ax}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda Vx}=(\boldsymbol{Vx})^{\mathrm{T}}\boldsymbol{\Lambda Vx}=(\boldsymbol{e}^i)^{\mathrm{T}}\boldsymbol{\Lambda e}^i=\lambda^i<0 \tag{4.58}$$

这与$\boldsymbol{A}$半正定矛盾，所以半正定矩阵的所有特征值都非负。同样可以证明正定矩阵的所有特征值都为正，半负定矩阵的所有特征值都非正，以及负定矩阵的所有特征值都为负。半正定矩阵可以分解成两个矩阵的乘积：

$$\boldsymbol{A}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\lambda^1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \lambda^n\end{pmatrix}\boldsymbol{V}=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\sqrt{\lambda^1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sqrt{\lambda^n}\end{pmatrix}\begin{pmatrix}\sqrt{\lambda^1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sqrt{\lambda^n}\end{pmatrix}\boldsymbol{V}$$

$$=\boldsymbol{V}^{\mathrm{T}}\left(\boldsymbol{\Lambda}^{\frac{1}{2}}\right)^{\mathrm{T}}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{V}=\left(\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{V}\right)^{\mathrm{T}}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{V} \tag{4.59}$$

现在考虑半正定矩阵的二次型的最大值问题。如果限制$\boldsymbol{x}$是单位向量，则半正定矩阵$\boldsymbol{A}$的二次型$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$的最大值是$\boldsymbol{A}$的最大特征值$\lambda^1$，达到该最大值的$\boldsymbol{x}$是$\lambda^1$的某个单位特征向量：

$$\max_{\|\boldsymbol{x}\|=1}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=\lambda^1 \tag{4.60}$$

为证明式（4.60）的结论，首先考察对于正交矩阵$\boldsymbol{V}^{\mathrm{T}}$和任意单位向量$\boldsymbol{x}$，$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}$的模是多少：

$$\|\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}\|^2=(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x})^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}=\|\boldsymbol{x}\|^2 \tag{4.61}$$

从式（4.61）看出，用正交矩阵乘任意向量不改变该向量的长度。从几何角度的理解是：正交矩阵的行是标准正交基，用正交矩阵乘一个向量是将该向量投影在新的标准坐标系中，这相当于旋转该向量，不改变其长度。

$\boldsymbol{\Lambda}$和$\boldsymbol{V}^{\mathrm{T}}$是将半正定矩阵$\boldsymbol{A}$谱分解得到的对角阵和正交阵。对于任意单位向量$\boldsymbol{x}$，它可以被$\boldsymbol{V}^{\mathrm{T}}$的列线性表出$\boldsymbol{x}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}'$，于是$\boldsymbol{x}'=\boldsymbol{V}\boldsymbol{x}$：

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{x}=(\boldsymbol{x}')^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{x}'=\sum_{i=1}^{n}\lambda^i(x_i')^2 \tag{4.62}$$

根据之前的论证，$\boldsymbol{x}'$的模与$\boldsymbol{x}$的模相同，都为 1。在$\sum_{i=1}^{n}(x_i')^2=1$的限制下，当$\boldsymbol{x}'=\boldsymbol{e}^1$时，式（4.62）达到最大值$\lambda^1$。如果$\boldsymbol{V}\boldsymbol{x}=\boldsymbol{e}^1$，则$\boldsymbol{x}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{e}^1=\boldsymbol{v}^1$。所以当$\boldsymbol{x}$是$\boldsymbol{A}$的最大特征值$\lambda^1$的单位特征向量$\boldsymbol{v}^1$时，$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$取得最大值$\lambda^1$。

寻找模为 1 的限制下第二大的$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$值，是没有意义的。因为二次型是连续的，当$\boldsymbol{x}$无限接近$\boldsymbol{v}^1$时，$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$可以无限接近$\lambda^1$。所以我们加一个限制条件：$\boldsymbol{x}$模为 1 且与$\boldsymbol{v}^1$正交。寻找在这个限制下，$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$的最大值。这时有结论：

$$\max_{\|\boldsymbol{x}\|=1,\boldsymbol{x}\perp\boldsymbol{v}^1}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=\lambda^2 \tag{4.63}$$

在$\boldsymbol{x}$模为 1 且正交于$\boldsymbol{v}^1$的限制下，$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$能达到的最大值是$\boldsymbol{A}$的第二大特征值$\lambda^2$。当$\boldsymbol{x}$是$\lambda^2$的单

位特征向量$\boldsymbol{v}^2$时，$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$达到此值。我们证明这个结论：令$\boldsymbol{x}'=\boldsymbol{V}\boldsymbol{x}$，于是 $\boldsymbol{x}'$的第一个元素是$\boldsymbol{v}^1$与$\boldsymbol{x}$的内积。$\boldsymbol{x}$正交于$\boldsymbol{v}^1$的限制使$\boldsymbol{x}'$的第一个分量为 0。根据式（4.62），欲使$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$达到最大，就必须把所有“份额”都分配给$\boldsymbol{x}'$的第 2 个分量，即$\boldsymbol{x}'=\boldsymbol{e}^2$，这时$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$达到限制下的最大值$\lambda^2$，有$\boldsymbol{x}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{e}^2=\boldsymbol{v}^2$。

我们总结一下：半正定矩阵的最大特征值的单位特征向量，在模为 1 的限制下使二次型$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$达到最大值，为最大特征值；第二大特征值的单位特征向量，在模为 1 且正交于最大特征值的单位特征向量的限制下，使二次型$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$达到最大值，为第二大特征值，以此类推。

4.2　多元函数的局部二阶特性

4

函数的局部一阶特性由梯度描述，梯度决定法向量，法向量决定切平面。切平面所能包含的信息无非是向何方倾斜，倾斜程度如何，这些信息就是函数向各方向的方向导数。如果切平面不倾斜，即梯度为零向量，这样的点就是驻点，驻点向各个方向的方向导数都为 0。但仅根据一阶信息无法判断驻点的类型，若要进一步刻画函数的局部形态，就需要用二次函数对原函数进行局部近似。函数的局部二次近似信息包含在赫森矩阵之中。

4.2.1　赫森矩阵

n元函数$f(\boldsymbol{x})$在$\boldsymbol{x}$点的赫森（Hessian）矩阵是$n\times n$矩阵$\boldsymbol{H}(\boldsymbol{x})$：

$$\boldsymbol{H}(\boldsymbol{x})=\begin{pmatrix}\frac{\partial f(\boldsymbol{x})}{\partial x_1\partial x_1} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n\partial x_1}\\ \vdots & \ddots & \vdots\\ \frac{\partial f(\boldsymbol{x})}{\partial x_1\partial x_n} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n\partial x_n}\end{pmatrix} \tag{4.64}$$

$\frac{\partial f(\boldsymbol{x})}{\partial x_j\partial x_i}$表示将$f(\boldsymbol{x})$先对$x_i$求偏导，再对$x_j$求偏导，即$f(\boldsymbol{x})$在$\boldsymbol{x}$的二阶偏导。有一个结论本书不给出证明：如果$\frac{\partial f(\boldsymbol{x})}{\partial x_j\partial x_i}$和$\frac{\partial f(\boldsymbol{x})}{\partial x_i\partial x_j}$在$\boldsymbol{x}$点都是连续的，则它们相等。也就是说，如果$f(\boldsymbol{x})$有连续的二阶偏导数，则求偏导的次序不影响结果。本书假设这种连续性都是满足的，所以$\boldsymbol{H}(\boldsymbol{x})$是对称矩阵。赫森矩阵$\boldsymbol{H}(\boldsymbol{x})$的第一行的$n$个元素分别是$\frac{\partial f(\boldsymbol{x})}{\partial x_1}$对$x_1,x_2,\cdots,x_n$的偏导，即$\boldsymbol{H}(\boldsymbol{x})$的第一行是$\frac{\partial f(\boldsymbol{x})}{\partial x_1}$的梯度的转置。

4.2.2　二阶泰勒展开

函数$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\boldsymbol{d}$的方向导数是$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$。当把自变量限制在一条直线上时，可以把$f(\boldsymbol{x})$看作一元函数，方向导数就是这个一元函数的导数。还可以定义$f(\boldsymbol{x})$沿$\boldsymbol{d}$的二阶导数，即方向导数的导数。将$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$视作$\boldsymbol{x}$的函数，求它沿$\boldsymbol{d}$的方向导数。首先求$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$的梯度：

$$\frac{\partial \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}}{\partial x_i} = \frac{\partial \sum_{j=1}^{n} d_j \frac{\partial f(\boldsymbol{x})}{\partial x_j}}{\partial x_i} = \sum_{j=1}^{n} d_j \frac{\partial f}{\partial x_i \partial x_j} = \boldsymbol{h}_{*,i}{}^{\mathrm{T}}\boldsymbol{d} \tag{4.65}$$

其中，$\boldsymbol{h}_{*,i}$是赫森矩阵$\boldsymbol{H}(\boldsymbol{x})$的第$i$列。式（4.65）说明$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$对$x_i$的偏导数是$\boldsymbol{H}(\boldsymbol{x})$的第$i$列与$\boldsymbol{d}$的内积，于是$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$在$\boldsymbol{x}$的梯度是：

$$\nabla(\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}) = \begin{pmatrix} \frac{\partial \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}}{\partial x_1} \\ \vdots \\ \frac{\partial \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \boldsymbol{h}_{*,1}{}^{\mathrm{T}}\boldsymbol{d} \\ \vdots \\ \boldsymbol{h}_{*,n}{}^{\mathrm{T}}\boldsymbol{d} \end{pmatrix} = \boldsymbol{H}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d} \tag{4.66}$$

所以方向导数$\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}$沿$\boldsymbol{d}$的方向导数，即$f(\boldsymbol{x})$沿$\boldsymbol{d}$的二阶导数是：

$$\left(\nabla(\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d})\right)^{\mathrm{T}}\boldsymbol{d} = (\boldsymbol{H}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d})^{\mathrm{T}}\boldsymbol{d} = \boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})\boldsymbol{d} \tag{4.67}$$

式（4.67）表明，$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\boldsymbol{d}$的二阶导数是二次型$\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})\boldsymbol{d}$。多元函数在某点沿各个方向的二阶导数全蕴含在赫森矩阵之中。回忆一下一元函数的二阶泰勒展开：

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + \mathcal{R}(h) \tag{4.68}$$

$\mathcal{R}(h)$是h^2的高阶无穷小，即：

$$\lim_{h\to 0}\frac{\mathcal{R}(h)}{h^2} = 0 \tag{4.69}$$

现在将二阶泰勒展开扩展到多元函数：

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h} + \frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})\boldsymbol{h}}{2} + \mathcal{R}(\boldsymbol{h}) \tag{4.70}$$

$\nabla f(\boldsymbol{x})$是$f(\boldsymbol{x})$在$\boldsymbol{x}$的梯度，$\boldsymbol{H}(\boldsymbol{x})$是$f(\boldsymbol{x})$在$\boldsymbol{x}$的赫森矩阵，$\mathcal{R}(\boldsymbol{h})$是$\|\boldsymbol{h}\|^2$的高阶无穷小：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|^2} = 0 \tag{4.71}$$

我们证明式（4.70），构造一元函数$g(h) = f\left(\boldsymbol{x} + h\cdot\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}\right)$，$\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$是单位向量。显然$f(\boldsymbol{x}+\boldsymbol{h}) = g(\|\boldsymbol{h}\|)$。$g(h)$在 0 点的导数是$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$的方向导数，$g(h)$在 0 点的二阶导数是$f(\boldsymbol{x})$在$\boldsymbol{x}$沿$\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$的二阶导数。把$g(h)$在 0 点二阶泰勒展开：

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{h}) &= g(\|\boldsymbol{h}\|) = g(0) + g'(0)\|\boldsymbol{h}\| + \frac{g''(0)}{2}\|\boldsymbol{h}\|^2 + \mathcal{R}(\|\boldsymbol{h}\|) \\ &= f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h} + \frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})\boldsymbol{h}}{2} + \mathcal{R}(\|\boldsymbol{h}\|) \end{aligned} \tag{4.72}$$

不论$\frac{h}{\|\boldsymbol{h}\|}$朝向什么方向，当$\|\boldsymbol{h}\|$趋近于 0 时，式(4.72)中的余项$\mathcal{R}(\|\boldsymbol{h}\|)$也趋近于 0，而且比$\|\boldsymbol{h}\|^2$消失得更快，这就证明了式（4.70）。对式（4.70）做一个变量替换，令$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{h}$，并省略高阶无穷小的余项，可得到原函数在$\boldsymbol{x}$附近的近似二次函数：

$$f(\boldsymbol{x}') \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{x}' - \boldsymbol{x}) + \frac{(\boldsymbol{x}' - \boldsymbol{x})^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})(\boldsymbol{x}' - \boldsymbol{x})}{2} \tag{4.73}$$

图 4-1 展示了几个典型的二次函数图像。后文我们将会看到：二次函数的赫森矩阵是常矩阵，它决定二次函数的形态。

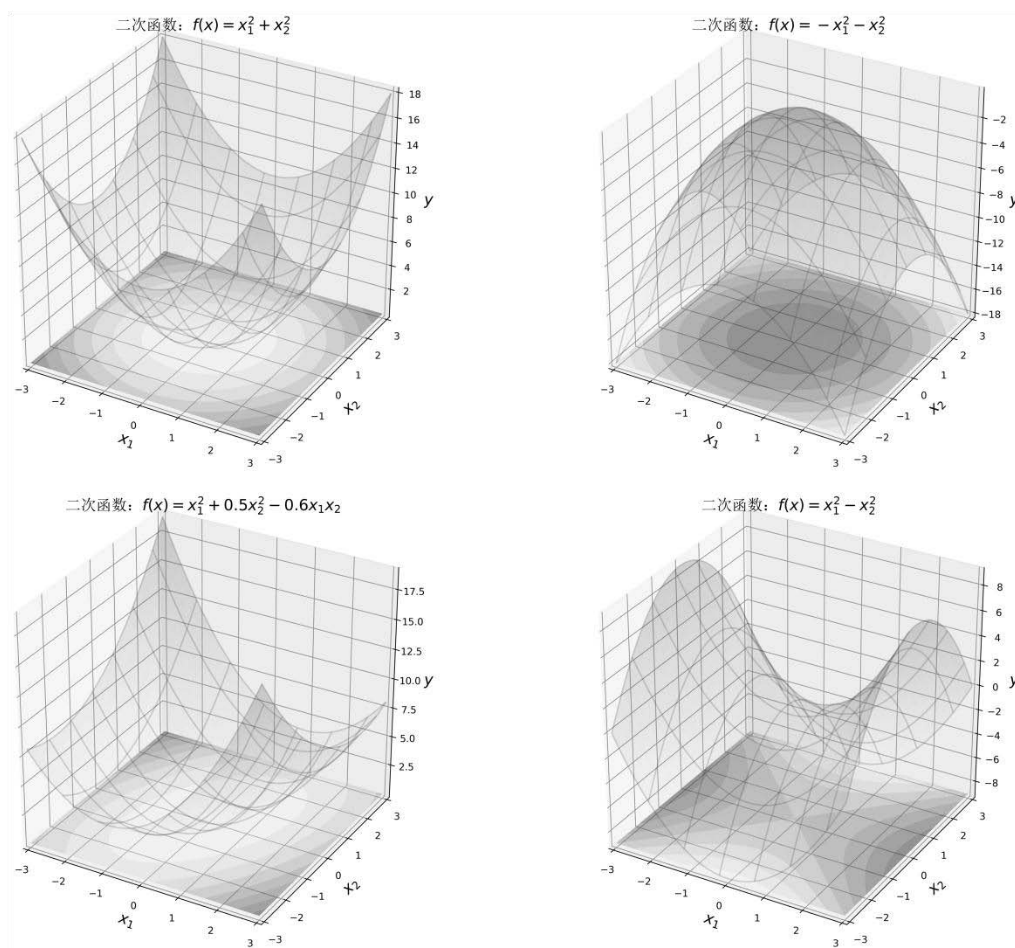


图 4-1　典型的二次函数图像

式（4.73）中的近似二次函数的图像过$(\boldsymbol{x}, f(\boldsymbol{x}))$点，它是$f(\boldsymbol{x})$在$\boldsymbol{x}$附近的二次近似。二次近似

函数的 0 次和一次项构成仿射函数$f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{x}'-\boldsymbol{x})$，这是前文讨论过的函数在$\boldsymbol{x}$附近的线性近似。二次近似是对原函数的更精确的近似，如图 4-2 所示。

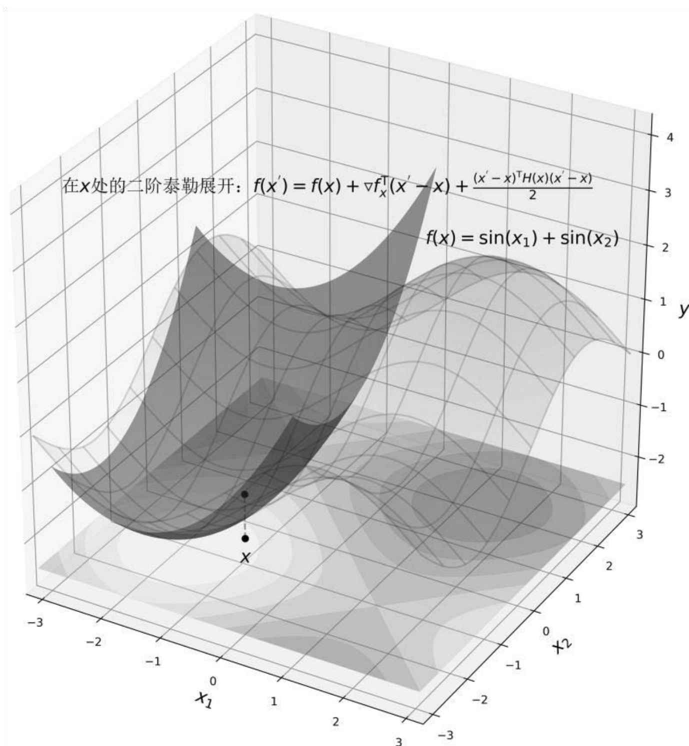


图 4-2　函数的局部二次近似

4.2.3　驻点的类型

驻点到底是局部极小点、局部极大点还是鞍点，这取决于函数的局部二阶特性。假设$\boldsymbol{x}^*$是$f(\boldsymbol{x})$的驻点，$\boldsymbol{H}(\boldsymbol{x}^*)$是$f(\boldsymbol{x})$在$\boldsymbol{x}^*$的赫森矩阵，因为$\nabla f(\boldsymbol{x}^*)$是零向量，$f(\boldsymbol{x})$在$\boldsymbol{x}^*$的二阶泰勒展开是：

$$f(\boldsymbol{x}^*+\boldsymbol{h})=f(\boldsymbol{x}^*)+\frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{h}}{2}+\mathcal{R}(\boldsymbol{h}) \tag{4.74}$$

令$\boldsymbol{d}=\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$，是与$\boldsymbol{h}$同方向的单位向量。假如$\boldsymbol{H}(\boldsymbol{x}^*)$是正定的，有$\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{d}>0$。因为$\mathcal{R}(\boldsymbol{h})$是$\|\boldsymbol{h}\|^2$的高阶无穷小，所以当$\|\boldsymbol{h}\|$趋近于 0 时，式（4.74）右边后两项有：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{h}/2+\mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|^2}=\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{d}>0 \tag{4.75}$$

当$\boldsymbol{x}+\boldsymbol{h}$足够靠近$\boldsymbol{x}$时，必有$f(\boldsymbol{x}^*+\boldsymbol{h})>f(\boldsymbol{x}^*)$，所以$\boldsymbol{x}^*$是局部极小点。注意，赫森矩阵必须

正定。如果赫森矩阵仅仅是半正定，那么式（4.75）的极限可能等于 0，后两项可以从负侧接近 0，这就不能保证$f(\boldsymbol{x}^* + \boldsymbol{h})$终会大于$f(\boldsymbol{x}^*)$。类似地，如果$\boldsymbol{H}(\boldsymbol{x}^*)$负定，则$\boldsymbol{x}^*$是局部极大点。

如果特征值有正也有负，则赫森矩阵是不定的。如果$\boldsymbol{H}(\boldsymbol{x}^*)$不定，则沿正特征值的特征向量的二阶导数为正，函数向上翘，驻点是最低点；沿负特征值的特征向量的二阶导数为负，函数向下弯，驻点是最高点。这种情况下，在任意小邻域内都有函数值大于$f(\boldsymbol{x}^*)$的点，也有函数值小于$f(\boldsymbol{x}^*)$的点，所以$\boldsymbol{x}^*$是鞍点。驻点类型与赫森矩阵的特征值的关系如图 4-3 所示。

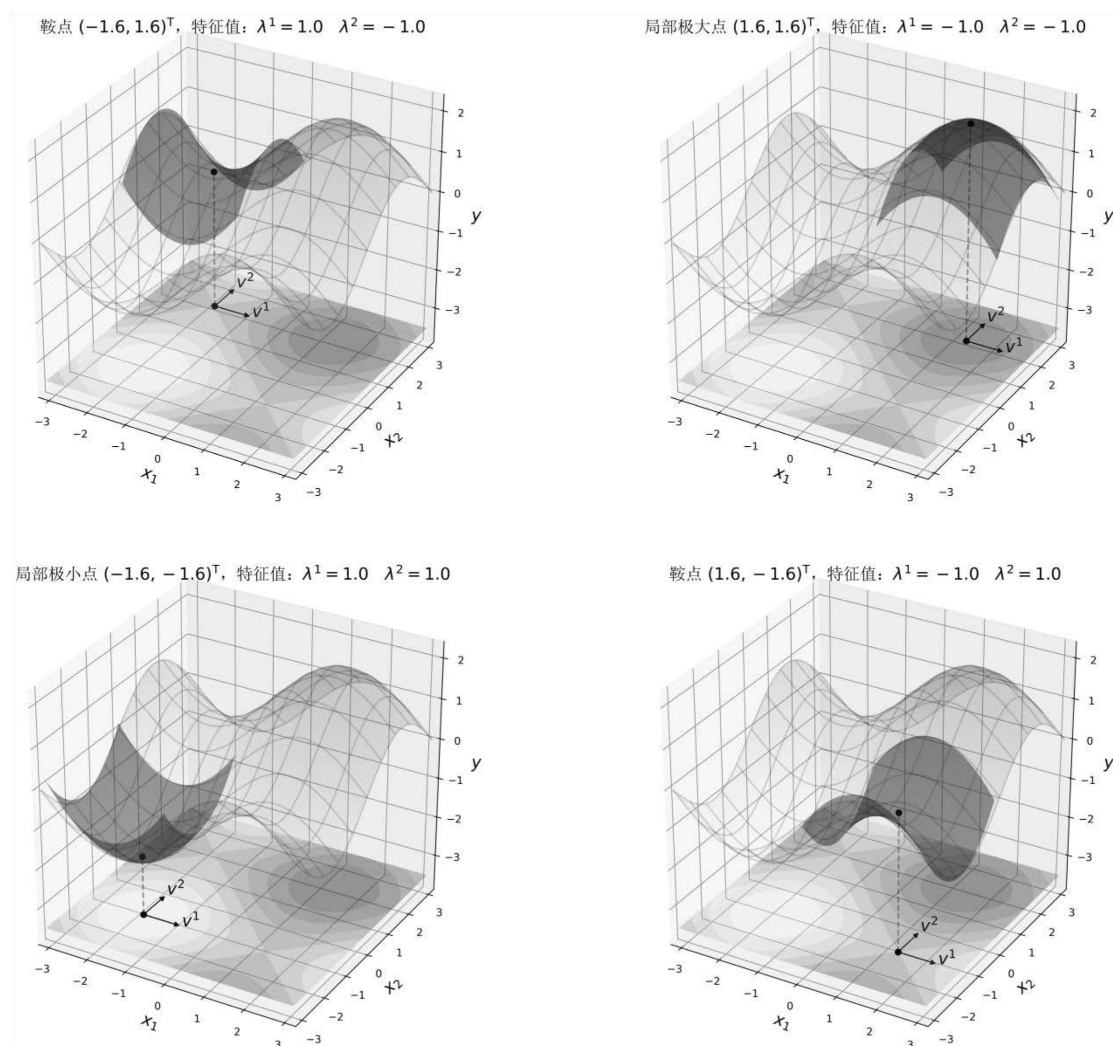


图 4-3　驻点类型与赫森矩阵的特征值的关系

我们总结一下驻点类型与赫森矩阵的关系。

- 驻点$\boldsymbol{x}^*$的赫森矩阵正定，$\boldsymbol{x}^*$是局部极小点；
- 驻点$\boldsymbol{x}^*$的赫森矩阵负定，$\boldsymbol{x}^*$是局部极大点；
- 驻点$\boldsymbol{x}^*$的赫森矩阵不定，$\boldsymbol{x}^*$是鞍点；
- 驻点$\boldsymbol{x}^*$的赫森矩阵半正定或半负定（但非正定或负定），$\boldsymbol{x}^*$的性质无法确定。

函数的局部二阶特性比一阶特性提供了更多的信息，但它仍不是函数的精确描述。如果赫森矩阵是奇异的，则它有 0 特征值，这时仍然没有足够的信息判断驻点的类型。如果不考虑奇异的情况，那么只有当赫森矩阵正定时，驻点是局部极小点。在没有任何先验知识时，可以假设特征值大于 0 的概率是 0.5 且彼此独立，如果模型有n个参数，则赫森矩阵的特征值全部大于 0 的概率是2^{-n}，这是一个极小的概率，所以在神经网络这种参数很多的模型中，局部极小点应该是极罕见的。

梯度反方向$-\nabla f$虽然是下降最快的方向，但如果沿$-\nabla f$的二阶导很大，则方向导数增大得很快。这种情况下，如果步长较大则可能导致函数值不降反升。如果在选择前进方向时不仅参考梯度$\nabla f(\boldsymbol{x})$，也参考赫森矩阵$\boldsymbol{H}(\boldsymbol{x})$，则算法将能做出更优的选择。后文将要介绍的牛顿法和共轭方向法就是如此。

4.2.4　赫森矩阵的条件数

令向量$\boldsymbol{x}=(x_1,x_2)^{\mathrm{T}}$，$q(\boldsymbol{x})$是二元二次函数：

$$q(\boldsymbol{x})=a+b_1x_1+b_2x_2+\frac{h_{1,1}}{2}{x_1}^2+\frac{h}{2}x_1x_2+\frac{h_{2,2}}{2}{x_2}^2=a+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} \tag{4.76}$$

其中，$\boldsymbol{b}=(b_1,b_2)^{\mathrm{T}}$，$\boldsymbol{H}=\begin{pmatrix}h_{1,1} & h_{1,2}\\ h_{2,1} & h_{2,2}\end{pmatrix}$，$h_{1,2}=h_{2,1}=\frac{h}{2}$。$q(\boldsymbol{x})$的梯度是：

$$\nabla q(\boldsymbol{x})=\begin{pmatrix}\frac{\partial q}{\partial x_1}\\ \frac{\partial q}{\partial x_2}\end{pmatrix}=\begin{pmatrix}b_1+h_{1,1}x_1+h_{1,2}x_2\\ b_2+h_{2,1}x_1+h_{2,2}x_2\end{pmatrix}=\boldsymbol{b}+\boldsymbol{H}\boldsymbol{x} \tag{4.77}$$

$q(\boldsymbol{x})$的赫森矩阵是：

$$\boldsymbol{H}=\begin{pmatrix}\frac{\partial q}{\partial x_1\partial x_1} & \frac{\partial q}{\partial x_2\partial x_1}\\ \frac{\partial q}{\partial x_1\partial x_2} & \frac{\partial q}{\partial x_2\partial x_2}\end{pmatrix}=\begin{pmatrix}h_{1,1} & h_{1,2}\\ h_{2,1} & h_{2,2}\end{pmatrix} \tag{4.78}$$

将$q(\boldsymbol{x}+\boldsymbol{h})$在$\boldsymbol{x}$二阶泰勒展开：

$$q(\boldsymbol{x}+\boldsymbol{h})=q(\boldsymbol{x})+(\boldsymbol{b}+\boldsymbol{H}\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h}+\frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{h}}{2}=a+\boldsymbol{b}^{\mathrm{T}}(\boldsymbol{x}+\boldsymbol{h})+\frac{(\boldsymbol{x}+\boldsymbol{h})^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}+\boldsymbol{h})}{2} \quad (4.79)$$

可见$q(\boldsymbol{x})$的二阶泰勒展开就是它本身。此结论不限于二元，n元二次函数的二阶泰勒展开就是它本身。因为二次函数的赫森矩阵是常矩阵，它的二阶特性是处处相同的。如果二次函数$q(\boldsymbol{x})$的赫森矩阵$\boldsymbol{H}$是正定的，则它的所有特征值为正，容易验证$\boldsymbol{H}$可逆，逆矩阵是：

$$\boldsymbol{H}^{-1}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}^{-1}\boldsymbol{V}=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\frac{1}{\lambda^1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\lambda^n}\end{pmatrix}\boldsymbol{V} \quad (4.80)$$

$\boldsymbol{V}^{\mathrm{T}}$是以$\boldsymbol{H}$的正交单位特征向量为列构成的矩阵。因为$\boldsymbol{H}$正定，所以$q(\boldsymbol{x})$的唯一驻点$\boldsymbol{x}^*=-\boldsymbol{H}^{-1}\boldsymbol{b}$是一个局部极小点。$q(\boldsymbol{x})$在$\boldsymbol{x}^*$沿任意方向的方向导数为 0，二阶方向导数是$\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}>0$，所以$q(\boldsymbol{x})$沿任意方向的方向导数持续增大。$q(\boldsymbol{x})$沿自变量空间中任意直线的图像都是先下降，在$\boldsymbol{x}^*$降到最低然后上升，是开口向上的抛物线。沿二阶导较大的方向抛物线较窄；沿二阶导较小的方向抛物线较宽。

根据之前对二次型最大值的讨论，使二阶导$\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}$最大的单位向量是$\boldsymbol{H}$的最大特征值$\lambda^1$的单位特征向量$\boldsymbol{v}^1$，在此方向上$q(\boldsymbol{x})$具有最大的二阶导数$\lambda^1$。垂直于$\boldsymbol{v}^1$且使二阶导$\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}$最大的单位向量是$\boldsymbol{H}$的第二大特征值$\lambda^2$的单位特征向量$\boldsymbol{v}^2$，在此方向上$q(\boldsymbol{x})$具有第二大的二阶导数$\lambda^2$。

$\boldsymbol{H}$的最大特征值与最小特征值之比$\frac{\lambda^{\max}}{\lambda^{\min}}$称为$\boldsymbol{H}$的条件数（condition number）。条件数越大，则函数图像的宽与窄相差越悬殊，图像越狭长。反之若$\frac{\lambda^{\max}}{\lambda^{\min}}=1$，则函数图像朝各个方向宽窄相同，呈圆碗状，如图 4-4 所示。

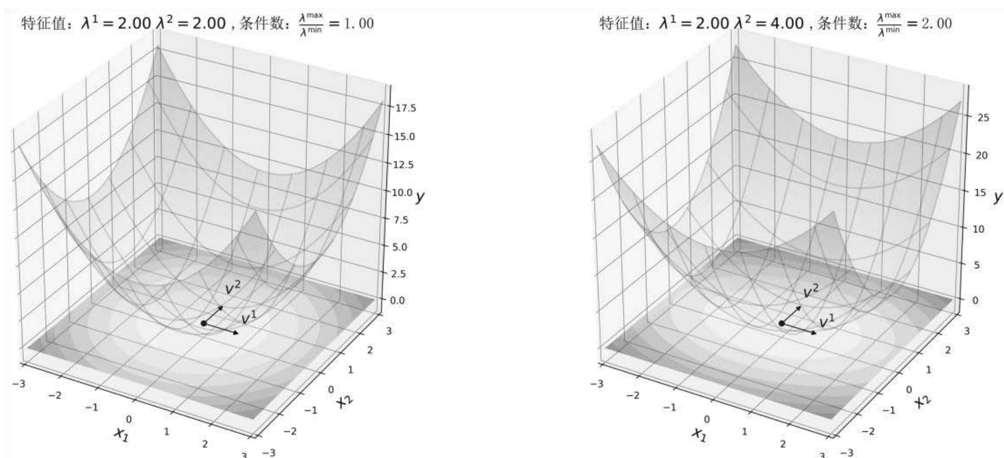


图 4-4　条件数与二次函数图像的狭长程度

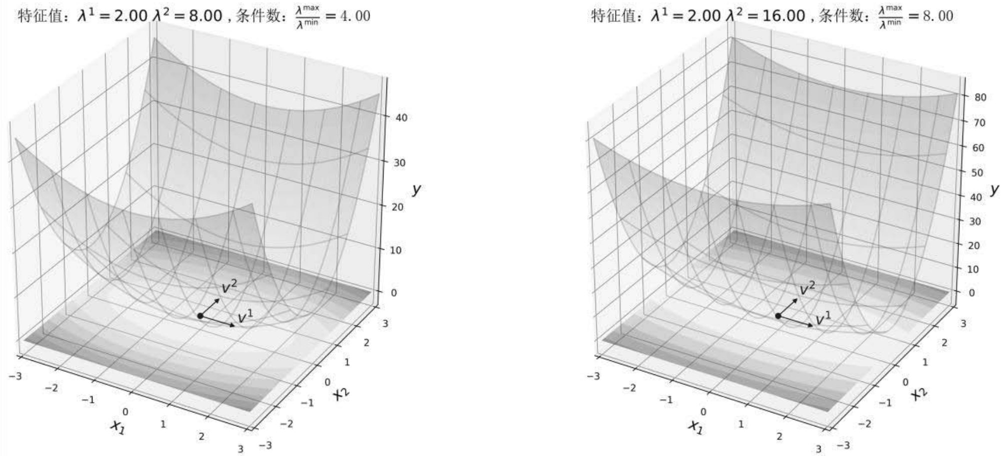


图 4-4　（续）

如果条件数$\frac{\lambda^{\max}}{\lambda^{\min}}$很大，则称该二次函数是病态的（ill-conditioned）。病态对梯度下降法不利，会引起震荡甚至不收敛。如果$\boldsymbol{H}$非常“健康”，它的条件数达到最小值 1，那么情况是怎么样的？如果$\boldsymbol{H}$的条件数是 1，即$\lambda^{\max} = \lambda^{\min}$，则$\boldsymbol{H}$的所有特征值都相等，令它们都等于$\lambda$，有：

$$\boldsymbol{H}^{-1} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}^{-1}\boldsymbol{V} = \boldsymbol{V}^{\mathrm{T}}\begin{pmatrix} \frac{1}{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\lambda} \end{pmatrix}\boldsymbol{V} = \frac{1}{\lambda}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \frac{1}{\lambda}\boldsymbol{I} \tag{4.81}$$

那么驻点$\boldsymbol{x}^*$是：

$$\boldsymbol{x}^* = -\boldsymbol{H}^{-1}\boldsymbol{b} = -\frac{1}{\lambda}\boldsymbol{b} \tag{4.82}$$

从任意点$\boldsymbol{x}$指向驻点$\boldsymbol{x}^*$的“箭头”是向量：

$$\boldsymbol{x}^* - \boldsymbol{x} = -\frac{1}{\lambda}\boldsymbol{b} - \boldsymbol{x} \tag{4.83}$$

再看看$\boldsymbol{x}$点的梯度反方向：

$$-\nabla q(\boldsymbol{x}) = -(\boldsymbol{b} + \boldsymbol{H}\boldsymbol{x}) = -(\boldsymbol{b} + \lambda\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{x}) = -(\boldsymbol{b} + \lambda\boldsymbol{x}) = \lambda(\boldsymbol{x}^* - \boldsymbol{x}) \tag{4.84}$$

式(4.84)表明，如果二次函数的赫森矩阵的条件数为 1，那么任一点$\boldsymbol{x}$的负梯度指向驻点$\boldsymbol{x}^*$。每一步梯度下降都是朝着正确的方向前进。当然如果步长过大有可能一步跨过$\boldsymbol{x}^*$而发生震荡，甚至有可能不收敛。现实中的损失函数不是二次函数，但是在局部可以用二次函数近似。本节分析的二次函数的特性，可以近似地表示函数的局部特性。

4.3 基于二阶特性的优化

如果参考函数的局部二阶信息，则优化算法会更健壮、更高效。但是高维情况下求赫森矩阵的计算量是巨大的，这妨碍了二阶优化算法在神经网络或深度学习中的应用。本节介绍两个基于赫森矩阵的优化算法——牛顿法和共轭方向法，并探讨它们的几何意义。从修正反梯度方向的角度看待牛顿法，可为上一章介绍的梯度下降法变体提供一些洞见。

4.3.1 牛顿法

牛顿法（Newton method）在某点对函数进行二阶泰勒展开，求函数的局部近似二次函数的驻点，以该驻点作为迭代的下一个点。根据式（4.70），$f(\boldsymbol{x})$在$\boldsymbol{x}$附近的局部二次近似函数是：

$$f(\boldsymbol{x}+\boldsymbol{h}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{h} + \frac{\boldsymbol{h}^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x})\boldsymbol{h}}{2} \tag{4.85}$$

二次近似的梯度是：

$$\nabla f(\boldsymbol{x}) + \boldsymbol{H}(\boldsymbol{x})\boldsymbol{h} \tag{4.86}$$

如果$\boldsymbol{H}(\boldsymbol{x})$的特征值都非 0（可逆），可用式（4.80）构造它的逆矩阵，再令梯度式（4.86）为零向量，求得驻点是：

$$\boldsymbol{h}^* = -\boldsymbol{H}(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x}) \tag{4.87}$$

所以，牛顿法对自变量$\boldsymbol{x}$的更新就是$\Delta\boldsymbol{x} = \boldsymbol{h}^* = -\boldsymbol{H}(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x})$。牛顿法的伪代码如下：

```
x⁰ ← 随机初始化
t ← 0
while ‖∇f(xᵗ)‖ ⩾ ε:
    xᵗ⁺¹ ← xᵗ − H(xᵗ)⁻¹∇f(xᵗ)
    t ← t + 1
return xᵗ
```

与梯度下降法一样，$\varepsilon > 0$是一个预设的阈值，当$\|\nabla f(\boldsymbol{x})\| < \varepsilon$时，认为$\nabla f(\boldsymbol{x})$已经足够接近零向量，算法停止。也可以采用循环次数达到预设的最大值，或者函数值的下降幅度小于阈值等其他停止标准。牛顿法迭代过程的示意图如图 4-5 所示。

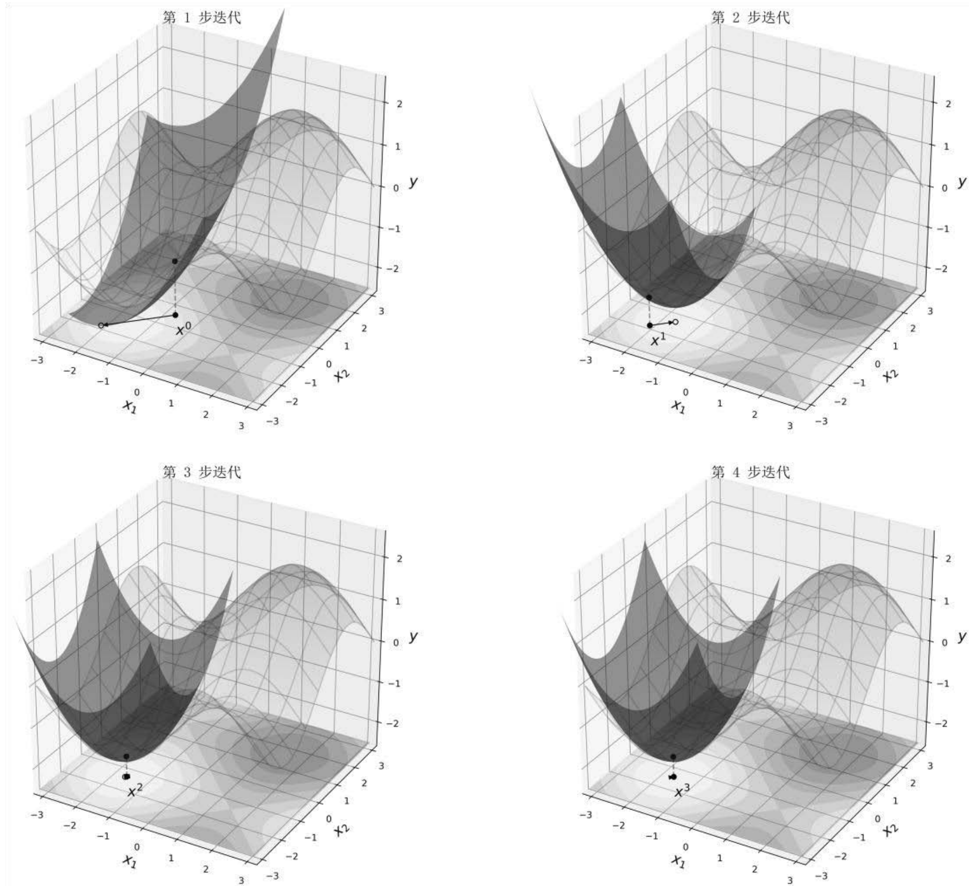


图 4-5　牛顿法迭代过程示意图

我们可以从对反梯度方向进行修正的角度看牛顿法。原始梯度下降法沿反梯度方向$-\nabla f(\boldsymbol{x})$更新自变量，而牛顿法将$-\nabla f(\boldsymbol{x})$乘上赫森矩阵的逆矩阵$\boldsymbol{H}(\boldsymbol{x})^{-1}$。假设$\boldsymbol{H}(\boldsymbol{x})$是正定的，根据对称矩阵的谱分解，$\boldsymbol{H}(\boldsymbol{x})^{-1}$可以分解成：

$$\boldsymbol{H}(\boldsymbol{x})^{-1}=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\frac{1}{\lambda^1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\lambda^n}\end{pmatrix}\boldsymbol{V} \tag{4.88}$$

$\boldsymbol{V}^{\mathrm{T}}$是由$\boldsymbol{H}(\boldsymbol{x})$的正交单位特征向量构成的正交矩阵，$\lambda^1,\lambda^2,\cdots,\lambda^n$是从大到小排列的$\boldsymbol{H}(\boldsymbol{x})$的$n$个正特征值（可重），于是牛顿法的更新式可以写成：

$$\Delta\boldsymbol{x}=-\boldsymbol{H}(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x})=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\frac{1}{\lambda^1} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\lambda^n}\end{pmatrix}\boldsymbol{V}\big(-\nabla f(\boldsymbol{x})\big) \tag{4.89}$$

$\boldsymbol{V}^{\mathrm{T}}$的列（即$\boldsymbol{V}$的行）是$\mathbb{R}^n$的标准正交基，也就是一个新的坐标系。该坐标系的第一个轴沿着$f(\boldsymbol{x})$二阶导数最大的方向，第二个轴沿着垂直于第一个轴且$f(\boldsymbol{x})$二阶导数次大的方向，以此类推。$\boldsymbol{V}\big(-\nabla f(\boldsymbol{x})\big)$的第一分量是反梯度向最大二阶导数方向的投影，第二分量是向次大二阶导数方向的投影，以此类推。

用对角阵乘$\boldsymbol{V}\big(-\nabla f(\boldsymbol{x})\big)$，相当于用权重$\frac{1}{\lambda^i}$惩罚各特征方向上$-\nabla f(\boldsymbol{x})$的分量。特征值$\lambda^i$越大，其特征方向上$f(\boldsymbol{x})$的二阶导数越大，对$-\nabla f(\boldsymbol{x})$在该方向上的分量的惩罚越严厉。病态条件下梯度下降法发生震荡的原因就是二阶导数较大的方向主宰了反梯度方向，牛顿法通过对二阶导数较大的方向进行适当惩罚缓解了这种现象，如图 4-6 所示。

二次函数的二阶泰勒展开就是它自身，牛顿法能够一步定位到二次函数的全局最小点。图 4-6 中的例子都是二次函数，可以看到图中修正后的向量都直指全局最小点。第一幅图中的二次函数的赫森矩阵的条件数为 1，反梯度本身就指向正确的方向。这时牛顿法缩放了反梯度向量的长度，使得修正后的向量直达全局最小点。注意，我们一直要求二次函数的赫森矩阵正定。因为如果赫森矩阵负定或不定，则二次函数没有全局最小点。

损失函数一般不是二次函数。在非二次函数上应用牛顿法，是在函数的局部二次近似上执行牛顿法。以当前点的局部二次近似的驻点作为下一个点进行迭代，希望找到全局最小点。和梯度下降法一样，在非二次函数上牛顿法不能保证找到全局最小点，但从上面的讨论可以看出，牛顿法修正了反梯度方向，在像“峡谷”这样的病态地形下能取得比梯度下降法更好的效果。

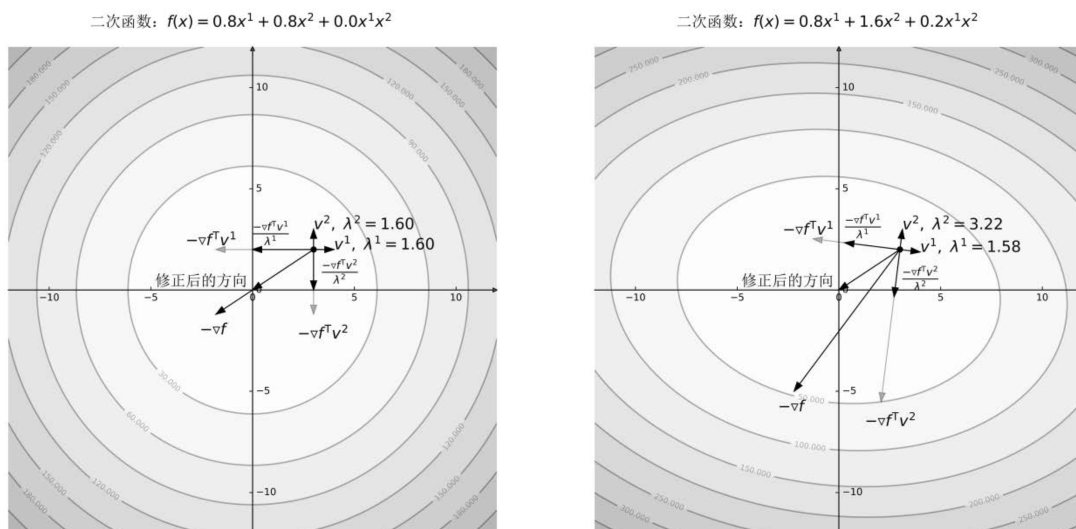


图 4-6　牛顿法对二阶导数较大的方向进行惩罚

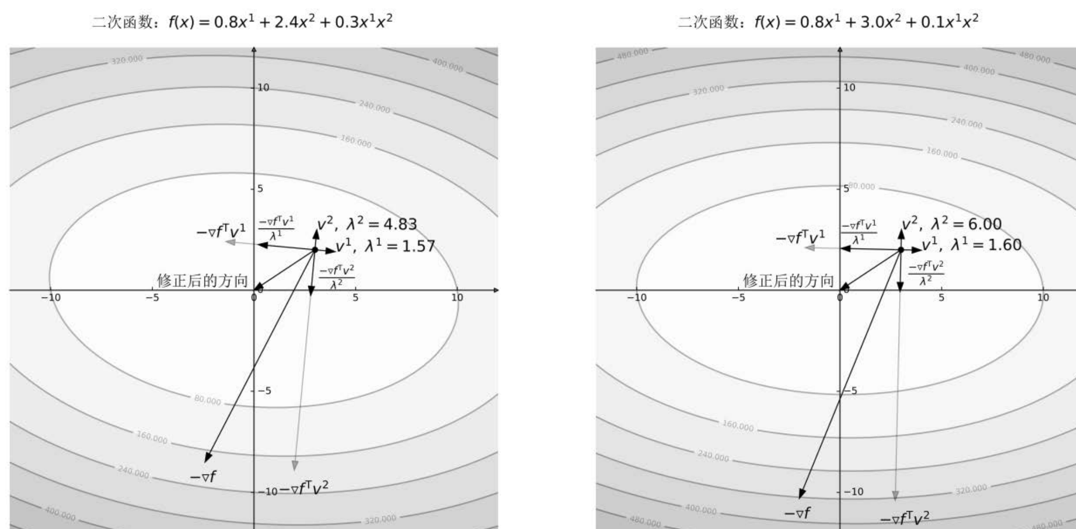


图 4-6　（续）

在非二次函数上应用牛顿法有一个问题，如果$f(\boldsymbol{x})$在$\boldsymbol{x}$的赫森矩阵非正定怎么办？例如，如果$f(\boldsymbol{x})$在$\boldsymbol{x}$的赫森矩阵是负定的，那么$f(\boldsymbol{x})$在$\boldsymbol{x}$附近的二次近似是一个倒扣的“碗”，牛顿法的更新公式定位到“倒扣碗”的驻点，那是近似二次函数的全局最大点，原函数在这一点很有可能有更高的函数值。从修正反梯度的角度看，如果赫森矩阵的特征值都是负值，那么$-\boldsymbol{H}(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x})$将$-\nabla f(\boldsymbol{x})$的每一个投影分量都取反了，修正后的方向指向的是函数值上升的方向，如图 4-7 所示。

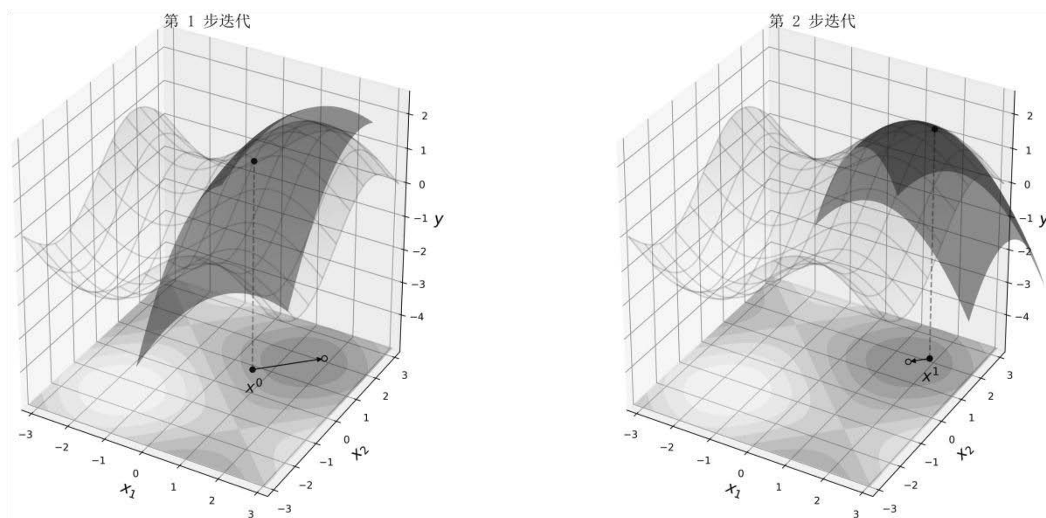


图 4-7　赫森矩阵负定导致牛顿法收敛到局部极大点

牛顿法有可能既到达不了局部极小点，也到达不了局部极大点，因为它的每一步是找局部二次近似的驻点，驻点还有一种可能性——鞍点。牛顿法有可能收敛到鞍点，如图 4-8 所示。

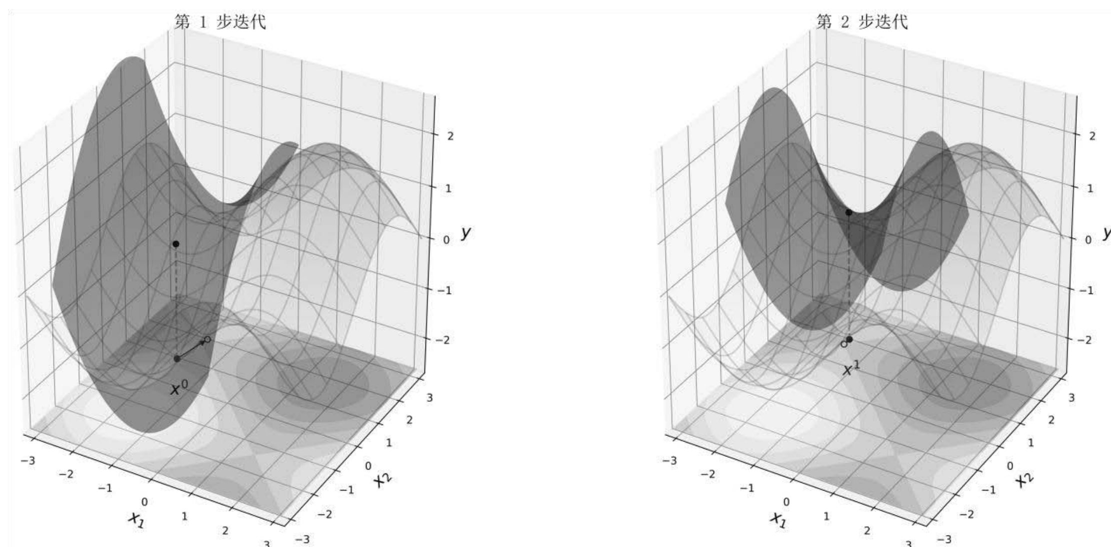


图 4-8　牛顿法收敛到鞍点

这一切都源于损失函数在各个点的赫森矩阵不一定正定。解决办法是对赫森矩阵进行修正，使它正定。例如 Levenberg-Marquardt 修正，以下简称 L-M 修正：

$$\widetilde{\boldsymbol{H}}(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x}) + \mu\boldsymbol{I} \tag{4.90}$$

如果λ^i（$i = 1,\cdots,n$）是$\boldsymbol{H}(\boldsymbol{x})$的特征值，$\boldsymbol{v}^i$（$i = 1,\cdots,n$）是相应的单位正交特征向量，那么$\lambda^i + \mu$（$i = 1,\cdots,n$）是$\widetilde{\boldsymbol{H}}(\boldsymbol{x})$的特征值，$\boldsymbol{v}^i$（$i = 1,\cdots,n$）仍是$\widetilde{\boldsymbol{H}}(\boldsymbol{x})$的相应特征向量，这是因为：

$$\widetilde{\boldsymbol{H}}(\boldsymbol{x})\boldsymbol{v}^i = (\boldsymbol{H}(\boldsymbol{x}) + \mu\boldsymbol{I})\boldsymbol{v}^i = \boldsymbol{H}(\boldsymbol{x})\boldsymbol{v}^i + \mu\boldsymbol{v}^i = \lambda^i\boldsymbol{v}^i + \mu\boldsymbol{v}^i = (\lambda^i + \mu)\boldsymbol{v}^i \tag{4.91}$$

如果μ足够大，则$\widetilde{\boldsymbol{H}}(\boldsymbol{x})$的特征值都将为正，$\widetilde{\boldsymbol{H}}(\boldsymbol{x})$正定。从修正反梯度方向的角度来考察：

$$\Delta\boldsymbol{x} = -\widetilde{\boldsymbol{H}}(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x}) = \boldsymbol{V}^{\mathrm{T}}\begin{pmatrix} \frac{1}{\lambda^1+\mu} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\lambda^n+\mu} \end{pmatrix}\boldsymbol{V}\left(-\nabla f(\boldsymbol{x})\right) \tag{4.92}$$

这同样是对反梯度各分量进行惩罚，惩罚因子是$\frac{1}{\lambda^i+\mu}$。当μ趋近于 0，L-M 修正趋近于原始牛顿法。当μ足够大，$\widetilde{\boldsymbol{H}}(\boldsymbol{x})^{-1}$的所有特征值都趋近于相同，L-M 修正趋近于梯度下降，所以 L-M 修正是在牛顿法和梯度下降法中间的一个折中。

如果模型参数很多，例如在神经网络和深度学习中常见的那样，计算赫森矩阵以及求逆的计算量不堪承受。有许多方法近似地模拟赫森矩阵或它的逆矩阵，它们被称为拟牛顿法（quasi-Newton method），本书不再展开介绍。

4.3.2 共轭方向法

共轭方向法的思想是在n维自变量空间中找到一组方向，依次沿这些方向寻找函数的最小点。当然，沿某一个方向的最小点并非全局最小点，但是这组方向具有一个性质：沿其中某一方向运动时可以保持函数值沿之前的方向为最小。当搜索过程进行到第n个方向，最终将找到全局最小值。这组方向必须关于赫森矩阵共轭。

二次函数$f(\boldsymbol{x})$的赫森矩阵是常矩阵$\boldsymbol{H}$，本节假设$\boldsymbol{H}$是正定的。在任意点$\boldsymbol{x}^*$附近将$f(\boldsymbol{x})$二阶泰勒展开：

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^*) \tag{4.93}$$

利用式（4.93）可以求$f(\boldsymbol{x})$在$\boldsymbol{x}$的梯度：

$$\nabla f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}^*) + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^*) \tag{4.94}$$

式（4.94）表明，$f(\boldsymbol{x})$在任意两点$\boldsymbol{x}$和$\boldsymbol{y}$的梯度之差$\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})$等于用赫森矩阵$\boldsymbol{H}$乘它们的差$\boldsymbol{x} - \boldsymbol{y}$。现在给出共轭的定义。如果单位向量$\boldsymbol{x}$和$\boldsymbol{y}$满足：

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} = 0 \tag{4.95}$$

则称$\boldsymbol{x}$和$\boldsymbol{y}$关于$\boldsymbol{H}$共轭（conjugate）。注意式（4.95）第一个等号是必然成立的，因为$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{y}$和$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}$是标量且互为转置，而标量的转置仍是自身。如果$\boldsymbol{H}$的所有特征值都相同，则$\boldsymbol{x}$和$\boldsymbol{y}$关于$\boldsymbol{H}$共轭等价于$\boldsymbol{x}$和$\boldsymbol{y}$正交：

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{y} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix} \lambda & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda \end{pmatrix}\boldsymbol{V}\boldsymbol{y} = \lambda\boldsymbol{x}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{y} = \lambda\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0 \tag{4.96}$$

共轭方向法在$\mathbb{R}^n$空间中寻找n个两两之间关于$\boldsymbol{H}$共轭的单位向量$\boldsymbol{d}^1, \boldsymbol{d}^2, \cdots, \boldsymbol{d}^n$，从任意一点$\boldsymbol{x}^0$开始沿直线$\boldsymbol{x}^0 + h\boldsymbol{d}^1$寻找$f(\boldsymbol{x})$的最小点$\boldsymbol{x}^1$，之后沿直线$\boldsymbol{x}^1 + h\boldsymbol{d}^2$寻找最小点$\boldsymbol{x}^2$，以此类推，直到找到$\boldsymbol{x}^n$。因为这组向量共轭，所以在每一条直线上搜索时可以保持在之前已经搜索过的方向上函数值仍为最小，最终找到的$\boldsymbol{x}^n$就是$f(\boldsymbol{x})$的全局最小点。后文会证明这一点，我们首先看看如何在直线$\boldsymbol{x} + h\boldsymbol{d}$上寻找使$f(\boldsymbol{x})$最小的点。令函数$g(h) = f(\boldsymbol{x} + h\boldsymbol{d})$，在$\boldsymbol{x}$点对$f(\boldsymbol{x} + h\boldsymbol{d})$泰勒展开，有：

$$g(h)=f(\boldsymbol{x}+h\boldsymbol{d})=f(\boldsymbol{x})+h\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+\frac{h^2}{2}\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d} \tag{4.97}$$

$g(h)$是关于h的一元二次函数，它对h的导数是：

$$\frac{\mathrm{d}g(h)}{\mathrm{d}h}=\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}+h\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d} \tag{4.98}$$

$g(h)$对h的二阶导数是：

$$\frac{\mathrm{d}^2g(h)}{\mathrm{d}h^2}=\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d} \tag{4.99}$$

因为之前假定$\boldsymbol{H}$正定，所以$g(h)$的二阶导数大于 0，于是$g(h)$的唯一驻点就是它的全局最小点。通过令式（4.98）为 0，求得$g(h)$的驻点是：

4

$$h^*=-\frac{\nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d}}{\boldsymbol{d}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}} \tag{4.100}$$

$g(h)$在h^*点的导数为 0，即$f(\boldsymbol{x})$在$\boldsymbol{x}^*=\boldsymbol{x}+h^*\boldsymbol{d}$沿$\boldsymbol{d}$的方向导数为 0：

$$\nabla_{\boldsymbol{d}}f(\boldsymbol{x}^*)=\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}\boldsymbol{d}=0 \tag{4.101}$$

共轭方向法从任意点$\boldsymbol{x}^0$开始，沿直线$\boldsymbol{x}^0+h\boldsymbol{d}^1$寻找最小点$\boldsymbol{x}^1$。根据式（4.100），有：

$$\boldsymbol{x}^1=\boldsymbol{x}^0-\frac{\nabla f(\boldsymbol{x}^0)^{\mathrm{T}}\boldsymbol{d}^1}{(\boldsymbol{d}^1)^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^1}\cdot\boldsymbol{d}^1 \tag{4.102}$$

另根据式（4.101），有$\nabla f(\boldsymbol{x}^1)^{\mathrm{T}}\boldsymbol{d}^1=0$。以此作为数学归纳法的起始条件，如果已经找到了$\boldsymbol{x}^k$，满足：

$$\nabla f(\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{d}^i=0,\quad i=1,\cdots,k \tag{4.103}$$

根据式（4.94），$f(\boldsymbol{x})$在直线$\boldsymbol{x}^k+h\boldsymbol{d}^{k+1}$上任意一点的梯度是：

$$\nabla f(\boldsymbol{x}^k+h\boldsymbol{d}^{k+1})=\nabla f(\boldsymbol{x}^k)+h\boldsymbol{H}\boldsymbol{d}^{k+1} \tag{4.104}$$

所以，$f(\boldsymbol{x})$在直线$\boldsymbol{x}^k+h\boldsymbol{d}^{k+1}$上沿之前$k$个方向$\boldsymbol{d}^i$（$i=1,\cdots,k$）的方向导数是：

$$\nabla f(\boldsymbol{x}^k+h\boldsymbol{d}^{k+1})^{\mathrm{T}}\boldsymbol{d}^i=\nabla f(\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{d}^i+h(\boldsymbol{d}^{k+1})^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^i=0,\quad i=1,\cdots,k \tag{4.105}$$

式（4.105）利用了$\boldsymbol{d}^{k+1}$与$\boldsymbol{d}^i$（$i=1,\cdots,k$）关于$\boldsymbol{H}$共轭。所以在直线$\boldsymbol{x}^k+h\boldsymbol{d}^{k+1}$上，$f(\boldsymbol{x})$沿之前$k$个方向的方向导数都为 0，也就是沿新方向搜索时可以保持在之前方向上函数值仍是最小。$f(\boldsymbol{x})$在直线$\boldsymbol{x}^k+h\boldsymbol{d}^{k+1}$上的最小点是：

$$\boldsymbol{x}^{k+1}=\boldsymbol{x}^k-\frac{\nabla f(\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{d}^{k+1}}{(\boldsymbol{d}^{k+1})^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^{k+1}}\cdot\boldsymbol{d}^{k+1} \tag{4.106}$$

且有$\nabla f(\boldsymbol{x}^{k+1})^{\mathrm{T}}\boldsymbol{d}^{k+1}=0$，这是因为$\boldsymbol{x}^{k+1}$是$f(\boldsymbol{x})$沿直线$\boldsymbol{x}^k+h\boldsymbol{d}^{k+1}$的最小点，$f(\boldsymbol{x})$在$\boldsymbol{x}^{k+1}$沿$\boldsymbol{d}^{k+1}$的方向导数为 0。利用数学归纳法就可以证明：从任意点出发依次沿着共轭方向寻找相应直线上的最小点，同时可以保持沿已经搜索过的方向的函数值仍为最小。

最后还需要证明最终的$\boldsymbol{x}^n$是整个$\mathbb{R}^n$空间中$f(\boldsymbol{x})$的全局最小点。如果一组向量$\boldsymbol{d}^1,\boldsymbol{d}^2,\cdots,\boldsymbol{d}^n$关于正定$\boldsymbol{H}$共轭，则它们一定线性独立，因为如果存在一组系数$w^1,w^2,\cdots,w^n$，满足$\sum_{i=1}^n w^i\boldsymbol{d}^i=0$，则对任意$j$有：

$$(\boldsymbol{d}^j)^{\mathrm{T}}\boldsymbol{H}\left(\sum_{i=1}^n w^i\boldsymbol{d}^i\right)=\sum_{i=1}^n w^i(\boldsymbol{d}^j)^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^i=w^j(\boldsymbol{d}^j)^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}^j=0 \tag{4.107}$$

式（4.107）利用了$\boldsymbol{d}^1,\boldsymbol{d}^2,\cdots,\boldsymbol{d}^n$两两共轭。因为$\boldsymbol{H}$正定，所以只能是$w^j$等于 0，这对任意$j$都成立，所以$\boldsymbol{d}^1,\boldsymbol{d}^2,\cdots,\boldsymbol{d}^n$线性独立。它们构成$\mathbb{R}^n$的一组基，所以任意一个单位向量$\boldsymbol{d}$都可以被$\boldsymbol{d}^1,\boldsymbol{d}^2,\cdots,\boldsymbol{d}^n$线性表出：$\boldsymbol{d}=\sum_{i=1}^n w^i\boldsymbol{d}^i$。于是$f(\boldsymbol{x})$在$\boldsymbol{x}^n$沿任意单位向量$\boldsymbol{d}$的方向导数是：

$$\nabla_{\boldsymbol{d}}f(\boldsymbol{x}^n)=\nabla f(\boldsymbol{x}^n)^{\mathrm{T}}\boldsymbol{d}=\sum_{i=1}^n w^i\nabla f(\boldsymbol{x}^n)^{\mathrm{T}}\boldsymbol{d}^i=0 \tag{4.108}$$

$f(\boldsymbol{x})$在$\boldsymbol{x}^n$沿任意方向的方向导数都为 0，所以$f(\boldsymbol{x})$在$\boldsymbol{x}^n$的梯度必然为零向量，于是$\boldsymbol{x}^n$是二次函数$f(\boldsymbol{x})$的驻点，也就是全局最小点。共轭方向法用n步找到二次函数的全局最小点。

还剩下一个问题，如何找到一组关于$\boldsymbol{H}$共轭的单位向量？回忆对称矩阵的谱分解，可以找到$\boldsymbol{H}$的一组正交单位特征向量$\boldsymbol{v}^1,\boldsymbol{v}^2,\cdots,\boldsymbol{v}^n$，对于任意$\boldsymbol{v}^i$和$\boldsymbol{v}^j$，有：

$$(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{H}\boldsymbol{v}^j=(\boldsymbol{v}^i)^{\mathrm{T}}(\lambda^j\boldsymbol{v}^j)=\lambda^j(\boldsymbol{v}^i)^{\mathrm{T}}\boldsymbol{v}^j=0 \tag{4.109}$$

所以$\boldsymbol{v}^1,\boldsymbol{v}^2,\cdots,\boldsymbol{v}^n$就是一组关于$\boldsymbol{H}$共轭的单位向量。共轭方向法的伪代码如下：

```
x^0 ← 随机初始化
v^1,v^2,⋯,v^n ← 赫森矩阵的单位正交特征向量
for t in 1 to n:
    x^t ← x^(t-1) − [∇f(x^(t-1))^T v^t / ((v^t)^T H v^t)] · v^t
return x^n
```

共轭方向不一定要取$\boldsymbol{v}^1,\boldsymbol{v}^2,\cdots,\boldsymbol{v}^n$，可以是任意一组共轭方向。对于非二次函数，可在某一点对它的局部二次近似执行共轭方向法，找到该二次近似的全局最小点，以该点作为迭代的下一个点。如果某一点的赫森矩阵非正定，可以采用诸如 L-M 修正对赫森矩阵进行修正。图 4-9 展示了在二元二次函数上执行共轭方向法的搜索路径。

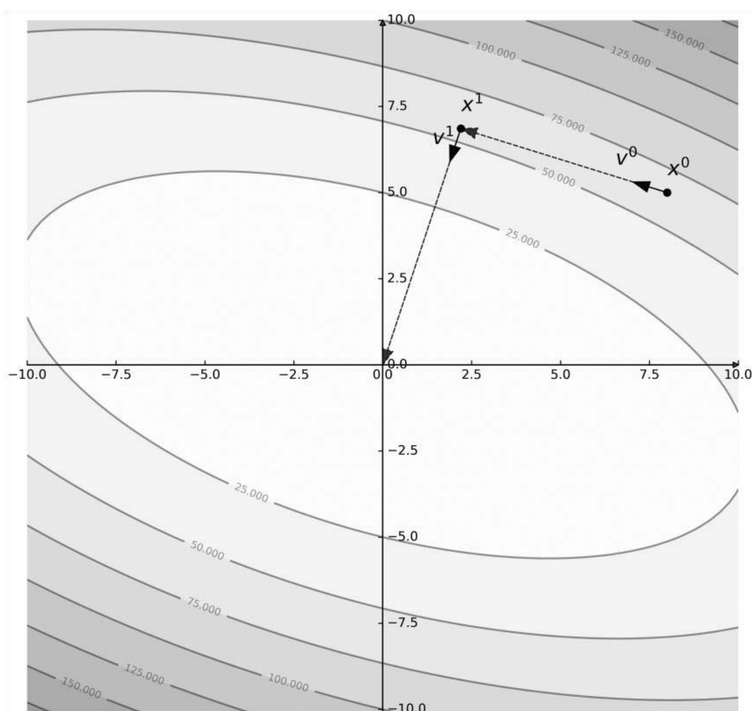


图 4-9 在二元二次函数上执行共轭方向法

4.4 运用牛顿法训练逻辑回归

本节介绍如何用牛顿法训练逻辑回归模型。为表示得简洁，先对逻辑回归的输入和参数进行一个处理，为输入向量增加一维常量 1，使之成为$n+1$维向量：

$$\boldsymbol{x}=\begin{pmatrix} x_0=1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{4.110}$$

将偏置b增加到权值向量，使它也成为$n+1$维向量：

$$\boldsymbol{w}=\begin{pmatrix} w_0=b \\ w_1 \\ \vdots \\ w_n \end{pmatrix} \tag{4.111}$$

通过添加一个常量输入 1，并将偏置b纳入权重向量$\boldsymbol{w}$，这样就可以认为没有偏置。于是逻辑回归的表达式可以写成：

$$f(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}} \tag{4.112}$$

第 3 章已经计算得到，交叉熵损失函数对权重向量$\boldsymbol{w}$的梯度是：

$$\nabla \text{loss}(\boldsymbol{w})=-\frac{1}{M}\sum_{s=1}^{M}\left(y^s-p^s(\boldsymbol{w})\right)\begin{pmatrix}x_0^s\\x_1^s\\\vdots\\x_n^s\end{pmatrix}=-\frac{1}{M}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{p}(\boldsymbol{w})\right) \tag{4.113}$$

训练集包含M个样本。$\boldsymbol{y}$是由全体训练样本的标签构成的M维向量。$\boldsymbol{p}(\boldsymbol{w})$是由全体预测概率构成的$M$维向量。$\boldsymbol{X}$是$M\times(n+1)$的矩阵，它的每一行是一个训练样本$\boldsymbol{x}^s$（$s=1,\cdots,M$）。$\boldsymbol{X}$称为设计矩阵（design matrix）。

损失函数$\text{loss}(\boldsymbol{w})$的赫森矩阵的第$i$行、第$j$列元素是其梯度$\nabla\text{loss}(\boldsymbol{w})$的第$i$分量对$w_j$的偏导数。根据式（4.113），$\nabla\text{loss}(\boldsymbol{w})$的第$i$分量是一个平均值，先计算其中一项对$w_j$的偏导数：

$$\frac{\partial\left(y^s-p^s(\boldsymbol{w})\right)x_i^s}{\partial w_j}=\frac{\partial\left(y^s-\frac{1}{1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^s}}\right)x_i^s}{\partial w_j}=-x_i^s x_j^s\frac{\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^s}}{\left(1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^s}\right)^2}=-x_i^s x_j^s p^s(\boldsymbol{w})\left(1-p^s(\boldsymbol{w})\right) \tag{4.114}$$

对式（4.114）求平均，得到赫森矩阵$\boldsymbol{H}(\boldsymbol{w})$的第$i$行、第$j$列元素是：

$$\boldsymbol{H}(\boldsymbol{w})_{i,j}=-\frac{1}{M}\sum_{s=1}^{M}\frac{\partial\left(y^s-p^s(\boldsymbol{w})\right)x_i^s}{\partial w_j}=\frac{1}{M}\sum_{s=1}^{M}x_i^s x_j^s p^s(\boldsymbol{w})\left(1-p^s(\boldsymbol{w})\right) \tag{4.115}$$

于是可以将$(n+1)\times(n+1)$的赫森矩阵$\boldsymbol{H}(\boldsymbol{w})$写为：

$$\boldsymbol{H}(\boldsymbol{w})=\frac{1}{M}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w})\boldsymbol{X} \tag{4.116}$$

其中，$\boldsymbol{X}$是$M\times(n+1)$的设计矩阵，$\boldsymbol{P}(\boldsymbol{w})$是$M\times M$的对角矩阵：

$$\boldsymbol{P}(\boldsymbol{w})=\begin{pmatrix}p^1(\boldsymbol{w})\left(1-p^1(\boldsymbol{w})\right) & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & p^M(\boldsymbol{w})\left(1-p^M(\boldsymbol{w})\right)\end{pmatrix} \tag{4.117}$$

现在我们有了损失函数$\text{loss}(\boldsymbol{w})$的梯度和赫森矩阵的逆矩阵$\boldsymbol{H}(\boldsymbol{w})^{-1}=M\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w})\boldsymbol{X}\right)^{-1}$。用牛顿法训练逻辑回归模型的伪代码如下：

```
w^0 ← 随机初始化
t ← 0
while ‖∇loss(w^t)‖ ⩾ ε:
    w^{t+1} ← w^t + (X^T P(w^t) X)^{-1} X^T (y − p(w^t))
    t ← t + 1
return w^t
```

现在抛开牛顿法的原理，从另一角度观察一下该更新式的含义。将算法迭代中的权值更新变形：

$$\boldsymbol{w}^t + (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\big(\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w}^t)\big) = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\Big(\boldsymbol{X}\boldsymbol{w}^t + \boldsymbol{P}(\boldsymbol{w}^t)^{-1}\big(\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w}^t)\big)\Big) \tag{4.118}$$

令$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}^t + \boldsymbol{P}(\boldsymbol{w}^t)^{-1}\big(\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w}^t)\big)$，于是算法更新式可以写成：

$$\boldsymbol{w}^{t+1} \leftarrow (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{z} \tag{4.119}$$

每一步迭代都将权值向量赋值为$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}(\boldsymbol{w}^t)\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}(\boldsymbol{w}^t)\boldsymbol{z}$。这是什么含义呢？首先看一下$\boldsymbol{z}$向量，它是当前仿射值$\boldsymbol{X}\boldsymbol{w}^t$加上一个加权误差：

$$\boldsymbol{P}(\boldsymbol{w}^t)^{-1}\big(\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w}^t)\big) = \begin{pmatrix} \frac{y^1 - p^1(\boldsymbol{w}^t)}{p^1(\boldsymbol{w}^t)\left(1 - p^1(\boldsymbol{w}^t)\right)} \\ \frac{y^2 - p^2(\boldsymbol{w}^t)}{p^2(\boldsymbol{w}^t)\left(1 - p^2(\boldsymbol{w}^t)\right)} \\ \vdots \\ \frac{y^M - p^M(\boldsymbol{w}^t)}{p^M(\boldsymbol{w}^t)\left(1 - p^M(\boldsymbol{w}^t)\right)} \end{pmatrix} \tag{4.120}$$

对于某个训练样本，模型输出的概率$p^s(\boldsymbol{w}^t)$越接近 1.0 或 0.0，则$p^s(\boldsymbol{w}^t)\big(1 - p^s(\boldsymbol{w}^t)\big)$越小，加在误差$y^s - p^s(\boldsymbol{w}^t)$上的权重越大，也就是模型越“肯定”，则对其失误的惩罚越严厉。将加权误差加到当前仿射值上，就得到期待仿射值向量$\boldsymbol{z}$。考虑如下加权最小二乘损失函数：

$$L(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{P}(\boldsymbol{w}^t)(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{z})\|^2 \tag{4.121}$$

式（4.121）以$p^s(\boldsymbol{w}^t)\big(1 - p^s(\boldsymbol{w}^t)\big)$作为每个分量的权重，衡量仿射值$\boldsymbol{X}\boldsymbol{w}$和期待仿射值$\boldsymbol{z}$之间的距离。对于某个训练样本，模型输出概率越接近 0.5，则权重越大，也就是对模型不太肯定的样本给予更大的重视。$L(\boldsymbol{w})$是$\boldsymbol{w}$的二次函数，其梯度是：

$$\nabla L(\boldsymbol{w}) = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{z}) \tag{4.122}$$

现在令$\nabla L(\boldsymbol{w}) = \boldsymbol{0}$，求得驻点是：

$$\boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{w}^t)\boldsymbol{z} \tag{4.123}$$

驻点$\boldsymbol{w}^*$正是式（4.119）赋给$\boldsymbol{w}^{t+1}$的值。所以牛顿法每一步更新的含义是以模型的确定性为权重，用误差$\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w}^t)$构造期待仿射值向量$\boldsymbol{z}$，之后以模型不确定性为权重，寻找使$\boldsymbol{X}\boldsymbol{w}$与$\boldsymbol{z}$之间距离最小的$\boldsymbol{w}^*$作为下一步迭代值。

4.5 牛顿法训练逻辑回归的 Python 实现

本节，我们实现运用牛顿法训练的逻辑回归模型。代码主体与第 3 章的相同，但是不再使用梯度下降优化器，而是在循环体中硬编码牛顿法更新公式，请看如下代码：

```
import numpy as np


class LogisticRegression_nt:

    def __init__(self, iterations = 10):

        self.iterations = iterations # 迭代次数

    def train(self, x, y):
        """
        x 为矩阵，形状是 n_samples * n_features，每一行为一个样本
        y 为矩阵，形状是 n_samples * 1，元素为样本的标签，正类为 1， 负类为 0
        """

        # 在 x 最前面添加一列常数 1，作为偏置值的输入，以简化公式
        x = np.mat(np.c_[[1.0] * x.shape[0], x])
        I = np.eye(x.shape[1]) # 构造一个单位矩阵

        # 根据 x 的列数（特征数）随机初始化权值，此时偏置值纳入了权值向量，相当于第一个权值
        # 权值向量为 n_features + 1 维向量，每个分量以 0 均值、0.01 标准差的正态分布初始化
        self.weights = np.mat(np.random.normal(0, 0.01, size=x.shape[1])).T

        for i in range(self.iterations):

            # 计算当前模型对训练集样本的输出
            p = self.predict(x, False)
            w = np.mat(np.diag(np.multiply(p, 1.0 - p).A1))

            # 牛顿法的更新式
            self.weights = self.weights + (x.T * w * x + 1e-10 * I).I * x.T * (y - p)

            # 评估当前模型并打印训练信息
            if i % 1 == 0:
                # 交叉熵损失
                cross_entropy = (-y.T * np.log(p) - (1.0 - y).T * np.log(1 - p)) / y.shape[0]

                # 正确率
                accuracy = np.sum(((p > 0.5).astype(np.int) == y).
```

```
                astype(np.int)) / y.shape[0]

            print("迭代: {:d}, 交叉熵: {:.6f}, 正确率: {:.2f}%".format(
                i + 1, cross_entropy[0, 0], accuracy * 100))


    def predict(self, x, augment = True):
        """
        预测函数。x 为矩阵, 形状是 n_samples * n_features, 每一行为一个样本
        augment 参数指示是否要在特征矩阵前添加一列常量 1
        """
        # 在 x 最前面添加一列常数 1, 作为偏置值的输入
        if augment:
            x = np.mat(np.c_[[1.0] * x.shape[0], x])

        a = -np.matmul(x, self.weights)
        a[a > 1e2] = 1e2  # 防止数值过大
        p = 1.0 / (1.0 + np.power(np.e, a))

        # 剪裁概率值, 保证其为合法的概率值
        p[p >= 1.0] = 1.0 - 1e-10
        p[p <= 0.0] = 1e-10

        return p
```

代码中的 LogisticRegression_nt 类就是运用牛顿法训练的逻辑回归模型，它不再需要优化器对象。训练的核心在循环体中的这一行代码：

```
self.weights = self.weights + (x.T * w * x + 1e-10 * I).I * x.T * (y - p)
```

这就是牛顿法的更新公式。其中，1e-10 * I 是对角线元素为小常量值的对角矩阵，它被加在赫森矩阵上，防止赫森矩阵奇异，这就是 L-M 修正。机器学习算法编程是背后数学原理的实现，与其他程序不同，仅仅看源代码无法明白程序是如何运作的。上述实现的关键之处只有这一句，必须理解原理才能明白它为什么起作用。现在将该模型用于鸟类生态类群问题。在这个问题下，牛顿法具有更快的收敛速度。

```
import pandas as pd
import numpy as np
from optimizer import *
from lr_nt import LogisticRegression_nt
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
```

4

```
bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)

# 根据标签是否属于"SW", "W", "R"三类，构造二分类 1/0 标签
bird["type"] = bird.type.apply(lambda t: t in ["SW", "W", "R"]).astype(np.int)

data = bird.values
# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集
train_x = np.mat(data[:300,:-1])
train_y = np.mat(data[:300,-1]).T

# 其余样本作为测试集
test_x = np.mat(data[300:,:-1])
test_y = np.mat(data[300:,-1]).T

# 构造逻辑回归对象，迭代次数取 10
lr = LogisticRegression_nt(10)

# 在训练集上训练
lr.train(train_x, train_y)

# 对测试集进行预测
p = lr.predict(test_x) # 模型预测的正类概率
pred = (p > 0.5).astype(np.int) # 以 0.5 为阈值时，模型预测的类别

print("正确率：{:.2f}%，查准率：{:.2f}%，查全率：{:.2f}%，ROC 曲线下面积：{:.3f}".format(
    accuracy_score(test_y, pred) * 100,
    precision_score(test_y, pred) * 100,
    recall_score(test_y, pred) * 100,
    roc_auc_score(test_y.A, p)))
```

4.6　小结

本章首先回顾了必要的矩阵知识，之后介绍了多元函数的局部二阶特性——赫森矩阵以及二阶泰勒展开。二阶泰勒展开是函数在局部的二次近似。二次函数的图像是抛物面——或椭圆抛物面或双曲抛物面（马鞍面）。无疑，二次函数的形状比平面更复杂，它能够更精确地近似函数的局部形状。如果优化算法能够利用损失函数的局部二阶信息，它就应该能够比梯度下降法表现得更好。

本章介绍了两个基于函数局部二阶特性的优化算法：牛顿法和共轭方向法。当待优化函数是二次函数且赫森矩阵正定时，它们分别能够以 1 步和n（自变量空间的维数）步找到全局最小点。

对于非二次函数，可在一个点的局部二次近似上施加牛顿法或共轭方向法，以二次近似的全局最小点作为迭代的下一个点。从修正反梯度方向的视角看牛顿法，我们发现它惩罚二阶导数较大的特征方向上的反梯度分量。当函数在各个特征方向上的二阶导数差异较大时，牛顿法所做的修正可避免较大二阶导数方向主导反梯度方向，从而一定程度上改善梯度下降法的效果。

当模型参数很多时，计算赫森矩阵及其逆矩阵的计算量是巨大的，所以二阶算法在神经网络和深度学习中并不常用，但是了解二阶优化算法对于理解损失函数的局部形态很有帮助。本章最后介绍了如何运用牛顿法训练逻辑回归模型，这对于深刻理解逻辑回归是一个有益的补充。阅读完本章，读者应该对函数的局部形态有了更透彻的理解，同时对函数优化也有了更深刻的洞见。

第5章 正 则 化

在机器学习建模实践中，模型自由度、偏置–方差权衡、过拟合与欠拟合以及正则化是很重要的概念。模型对未来样本的预测误差由三部分组成：模型预测的偏置、模型预测的方差以及问题固有的误差。模型预测的偏置与方差此消彼长，自由度控制模型在高偏置–低方差和低偏置–高方差两个极端之间进行权衡，这种权衡影响模型未来的预测表现。模型自由度受很多因素影响，正则化是控制自由度的一个重要手段。

本章首先回顾必要的概率论知识，之后我们介绍线性回归模型，并在线性回归的框架下讲解模型的自由度和偏置–方差权衡。在线性回归框架下，这些概念有精确的数学表述。接着，我们由“岭回归”而引入$\mathcal{L}_2$正则化，并从主成分和贝叶斯后验概率两种视角解释$\mathcal{L}_2$正则化的含义。我们还简单讲解$\mathcal{L}_1$正则化、它的效果以及它与$\mathcal{L}_2$正则化的异同。最后我们介绍如何在逻辑回归的训练中引入$\mathcal{L}_2$正则化。

这些概念虽然是在线性回归框架下引入和阐释的，但它们也适用于任何机器学习模型。$\mathcal{L}_1$和$\mathcal{L}_2$正则化可以直接应用于神经网络和深度学习。除正则化外，模型自由度还受很多其他因素影响，例如模型的结构和规模、对训练过程的限制、随机性等。建模实践中最重要的就是运用各种显式的超参数和隐含的设计选择来调节模型自由度，在偏置–方差权衡中寻找最佳模型。

5.1 概率论回顾

第 2 章介绍交叉熵损失的贝叶斯视角时，已经涉及了一部分概率论方面的内容，本章在进入正题之前还需要更系统地回顾一些概率论的知识。本节的回顾着重于多元随机变量，即随机向量。

5.1.1 随机变量

本节首先回顾一元随机变量的相关知识，包括概率密度、独立、期望、方差、协方差、相关

系数。本书用斜细体大写字母表示随机变量，如X和Y。对于一个随机变量X，存在一个连续函数$p_X: \mathbb{R} \to \mathbb{R}$，$X$落在区间$S \subseteq \mathbb{R}$的概率是：

$$P(X \in S) = \int_S p_X(x)\,\mathrm{d}x \tag{5.1}$$

$p_X(x)$的值称为随机变量X在x的概率密度。函数$p_X(x)$称为随机变量X的概率密度函数或分布，可简写为p_X。由于X落在任何区间的概率都不为负，所以对任何x必有$p_X(x) \geqslant 0$，否则如果对某个x^*有$p_X(x^*) < 0$，因为$p_X(x)$连续，所以存在x^*的一个邻域$S = (x^* - \epsilon, x^* + \epsilon)$，对$S$内的所有$x$，都有$p_X(x) < 0$，于是$\int_S p_X(x)\,\mathrm{d}x < 0$，这与概率非负矛盾。由于$X$必然落在$\mathbb{R}$中，所以$X \in \mathbb{R}$概率是 1，即：

$$P(X \in \mathbb{R}) = \int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x = 1 \tag{5.2}$$

概率密度函数必须连续、非负且在实数轴上积分为 1。两个随机变量X和Y，X为x且Y为y的概率密度函数$p_{X,Y}(x,y)$称为X和Y的联合概率密度函数或联合分布（joint distribution）。将联合分布$p_{X,Y}(x,y)$对Y的所有可能取值做积分，就得到X的概率密度函数：

$$p_X(x) = \int_{-\infty}^{\infty} p_{X,Y}(x,y)\,\mathrm{d}y \tag{5.3}$$

也可以对X的所有取值做积分，得到Y的分布p_Y。如果X和Y的联合分布$p_{X,Y}$满足：

$$p_{X,Y} = p_X p_Y \tag{5.4}$$

则称随机变量X和Y是独立的（independent）。随机变量X在分布p_X下的期望（exception）定义为：

$$E_{p_X}(X) = \int_{-\infty}^{\infty} x p_X(x)\,\mathrm{d}x \tag{5.5}$$

期望是随机变量所有可能取值以概率密度为权的加权均值。不引起混淆的情况下可将$E_{p_X}(X)$写成E_X。对于X和Y以及实数a和b，随机变量$aX + bY$的期望是：

$$E_{p_{X,Y}}(aX + bY) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(ax + by)p_{X,Y}\,\mathrm{d}x\,\mathrm{d}y = a\int_{-\infty}^{\infty} x \int_{-\infty}^{\infty} p_{X,Y}\,\mathrm{d}y\,\mathrm{d}x +$$

$$b\int_{-\infty}^{\infty} y \int_{-\infty}^{\infty} p_{X,Y}\,\mathrm{d}x\,\mathrm{d}y = a\int_{-\infty}^{\infty} x p_X \mathrm{d}x + b\int_{-\infty}^{\infty} y p_Y \mathrm{d}y = aE_X + bE_Y \tag{5.6}$$

对于常数C，$X + C$的期望是：

$$E_{p_X}(X + C) = E_X + C\int_{-\infty}^{\infty} p_X\,\mathrm{d}x = E_X + C \tag{5.7}$$

如果X和Y独立，有：

5

$$E_{p_{X,Y}}(XY) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyp_{X,Y}\,\mathrm{d}x\,\mathrm{d}y = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyp_X p_Y\,\mathrm{d}x\,\mathrm{d}y$$

$$= \int_{-\infty}^{\infty} xp_X \mathrm{d}x \cdot \int_{-\infty}^{\infty} yp_Y \mathrm{d}y = E_X E_Y \tag{5.8}$$

$E_{p_{X,Y}}(XY)$可简写为E_{XY}。X在分布$p_X(x)$下的方差（variance）定义为：

$$\mathrm{var}_{p_X}(X) = E_{p_X}\left(\left(X - E_{p_X}(X)\right)^2\right) = \int_{-\infty}^{\infty}\left(x - E_{p_X}(X)\right)^2 p_X(x)\,\mathrm{d}x \tag{5.9}$$

方差是X与期望$E_{p_X}(X)$之差的平方在分布$p_X(x)$下的期望，不引起混淆的情况下可写作var_X。方差反映的是随机变量偏离其均值的分散程度。X的标准差（standard deviation）是它的方差的平方根：

$$\mathrm{std}_X = \sqrt{\mathrm{var}_X} \tag{5.10}$$

随机变量X和Y的协方差（covariance）是：

$$\mathrm{cov}_{p_{X,Y}}(X,Y) = E_{p_{X,Y}}\left(\left(X - E_{p_X}(X)\right)\left(Y - E_{p_Y}(Y)\right)\right) \tag{5.11}$$

不引起混淆的情况下可将协方差写作$\mathrm{cov}_{X,Y}$。var_X其实就是$\mathrm{cov}_{X,X}$。容易看出$\mathrm{cov}_{X,Y} = \mathrm{cov}_{Y,X}$。将$\mathrm{cov}_{X,Y}$的表达式变形：

$$\mathrm{cov}_{X,Y} = E\left((X - E_X)(Y - E_Y)\right) = E(XY - XE_Y - YE_X + E_X E_Y) = E_{XY} - E_X E_Y \tag{5.12}$$

推导式（5.12）时注意E_X和E_Y不是随机变量。根据式（5.12）有：

$$\mathrm{var}_X = \mathrm{cov}_{X,X} = E_{X^2} - {E_X}^2 \tag{5.13}$$

式（5.13）在分解模型预测误差时将会用到。根据式（5.13），对于任意实数a和b有：

$$\mathrm{cov}_{aX,bY} = E_{abXY} - E_{aX}E_{bY} = ab(E_{XY} - E_X E_Y) = ab \cdot \mathrm{cov}_{X,Y} \tag{5.14}$$

根据式（5.14）有$\mathrm{var}_{aX} = a^2 \cdot \mathrm{var}_X$。对于随机变量$X$，$Y$和$Z$有：

$$\mathrm{cov}_{X,Y+Z} = E_{X(Y+Z)} - E_X E_{Y+Z} = E_{XY} + E_{XZ} - E_X E_Y - E_X E_Z = \mathrm{cov}_{X,Y} + \mathrm{cov}_{X,Z} \tag{5.15}$$

至此，我们得到期望、方差和协方差的若干计算定律，后文会频繁用到。随机变量X和Y之间的相关系数（correlation coefficient）定义为：

$$\mathrm{corr}_{X,Y} = \frac{\mathrm{cov}_{X,Y}}{\mathrm{std}_X \cdot \mathrm{std}_Y} \tag{5.16}$$

如果X和Y独立，根据式（5.8）有$E_{XY} = E_X E_Y$，再根据式（5.12），它们的协方差和相关系数为 0，即如果X和Y独立，则它们不相关。但是X和Y不相关并不意味着它们独立。相关系数衡量

两个随机变量呈线性关系的程度，两个不独立的随机变量之间也可以几乎没有线性关系，这导致它们之间相关系数很小，如图 5-1 所示。

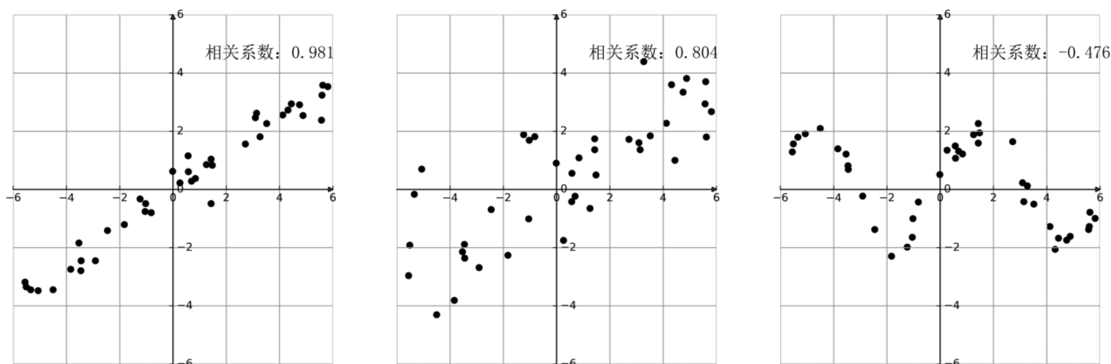


图 5-1 相关系数衡量两随机变量之间的线性关系

5.1.2 多元随机变量

多元随机变量即随机向量（random vector）。本书仍用斜细体大写字母表示随机向量，如X和Y。用脚标表示随机向量的分量，如X_i表示X的i分量。随机向量的取值是向量，如$\boldsymbol{x}$和$\boldsymbol{y}$。对随机向量X，存在连续的概率密度函数$p_X(\boldsymbol{x})$: $\mathbb{R}^n \to \mathbb{R}$，将$X$的取值$\boldsymbol{x} \in \mathbb{R}^n$映射到它的概率密度。$X$落在区域$\boldsymbol{S} \subseteq \mathbb{R}^n$的概率是：

$$P(X \in \boldsymbol{S}) = \int_{\boldsymbol{S}} p_X(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \int \int \cdots \int_{\boldsymbol{S}} p_X(\boldsymbol{x}) \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_n \quad (5.17)$$

对于任意$\boldsymbol{x}$，必须满足$p_X(\boldsymbol{x}) \geqslant 0$，并且$p_X(\boldsymbol{x})$必须满足：

$$P(X \in \mathbb{R}^n) = \int_{\mathbb{R}^n} p_X(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p(\boldsymbol{x})\, \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_n = 1 \quad (5.18)$$

随机向量X的第i分量X_i是一个随机变量。将$p_X(\boldsymbol{x})$对其他分量X_j（$j \neq i$）积分，可得到X_i的概率密度函数：

$$p_{X_i}(x) = \int_{\mathbb{R}^{n-1}} p_X(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}_{j \neq i} \quad (5.19)$$

其中，$p_{X_i}(x)$称为X的第i分量的边缘分布（marginal distribution）。同样可以求X的几个分量的联合边缘分布（joint marginal distribution），只要将$p_X(\boldsymbol{x})$在其他分量上积分即可。例如求第i和第j分量的联合边缘分布，就是将$p_X(\boldsymbol{x})$对其他分量X_s（$s \neq i, j$）做积分：

$$p_{X_i, X_j}(x) = \int_{\mathbb{R}^{n-2}} p_X(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}_{s \neq i, j} \quad (5.20)$$

5.1.3　多元随机变量的期望和协方差矩阵

随机向量X在分布$p_X(\boldsymbol{x})$下的期望是：

$$E_{p_X}(X)=\begin{pmatrix}E_{p_X}(X_1)\\ \vdots\\ E_{p_X}(X_n)\end{pmatrix}=\begin{pmatrix}\int_{\mathbb{R}^n}x_1p_X(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\\ \vdots\\ \int_{\mathbb{R}^n}x_np_X(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\end{pmatrix}=\begin{pmatrix}\int_{-\infty}^{\infty}x_1p_{X_1}(x)\mathrm{d}x\\ \vdots\\ \int_{-\infty}^{\infty}x_np_{X_n}(x)\mathrm{d}x\end{pmatrix}=\begin{pmatrix}E_{X_1}\\ \vdots\\ E_{X_n}\end{pmatrix}\quad(5.21)$$

式（5.21）的第三步变更多重积分的顺序，将求期望的分量提到最外层，内层积分的结果就是该分量的边缘概率密度。随机向量X的期望是由它的每个分量的期望组成的向量，可简写成E_X。随机向量X在分布$p_X(\boldsymbol{x})$下的协方差矩阵（covariance matrix）是：

$$\mathrm{cov}(X)=\begin{pmatrix}\mathrm{cov}_{X_1,X_1}&\cdots&\mathrm{cov}_{X_1,X_n}\\ \vdots&\ddots&\vdots\\ \mathrm{cov}_{X_n,X_1}&\cdots&\mathrm{cov}_{X_n,X_n}\end{pmatrix}\quad(5.22)$$

$\mathrm{cov}(X)$可简写作cov_X。它的第i行、第j列元素是分量X_i和X_j的协方差，对角线元素就是各分量的方差。因为$\mathrm{cov}_{X_i,X_j}=\mathrm{cov}_{X_j,X_i}$，所以$\mathrm{cov}_X$是对称矩阵。

容易验证，对于随机向量X和矩阵$\boldsymbol{A}$，$\boldsymbol{A}X$的期望$E_{\boldsymbol{A}X}$是$\boldsymbol{A}E_X$，$\boldsymbol{A}X$的协方差矩阵$\mathrm{cov}_{\boldsymbol{A}X}$是$\boldsymbol{A}\mathrm{cov}_{\boldsymbol{A}X}\boldsymbol{A}^{\mathrm{T}}$。只要将元素展开并计算它们的期望和协方差，就可以验证这两条性质，这里忽略细节。

5.1.4　样本均值和样本协方差矩阵

在实际问题中并不知道数据的真实分布，但可以从样本中估计数据的期望和协方差矩阵。假如观察到随机向量X的M个样本$\boldsymbol{x}^i$（$i=1,\cdots,M$），每个样本都是n维向量，将这些样本作为行，构造$M\times n$的设计矩阵：

$$\boldsymbol{X}_{M\times n}=\begin{pmatrix}(\boldsymbol{x}^1)^{\mathrm{T}}\\ (\boldsymbol{x}^2)^{\mathrm{T}}\\ \vdots\\ (\boldsymbol{x}^M)^{\mathrm{T}}\end{pmatrix}\quad(5.23)$$

样本均值$\overline{\boldsymbol{x}}$是全部样本的平均值：

$$\overline{\boldsymbol{x}}=\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}^i\quad(5.24)$$

令X^i（$i=1,\cdots,M$）是M个独立的随机向量，它们都与X同分布。$\bar{X}$是这M个随机向量的平均，也是一个随机向量。将每个样本$\boldsymbol{x}^i$视作对X^i的一次采样，则$\overline{\boldsymbol{x}}$就是对$\bar{X}$的一次采样。$\bar{X}$的期望是：

$$E_{\bar{X}} = E\left(\frac{1}{M}\sum_{i=1}^{M} X^i\right) = \frac{1}{M}\sum_{i=1}^{M} E(X^i) = \frac{1}{M}\sum_{i=1}^{M} E(X) = E_X \quad (5.25)$$

所以称$\bar{\boldsymbol{x}}$是E_X的无偏估计（unbiased estimator），因为$\bar{X}$的期望是E_X。现在估计X的第i和第j分量之间的协方差cov_{X_i,X_j}，我们定义样本协方差$s_{i,j}$是：

$$s_{i,j} = \frac{1}{M-1}\sum_{k=1}^{M}\left(x_i^k - \bar{x}_i\right)\left(x_j^k - \bar{x}_j\right) \quad (5.26)$$

x_i^k是第k个样本$\boldsymbol{x}^k$的第i分量，$\bar{x}_i$是全体样本的第i分量的平均值，x_j^k和$\bar{x}_j$也类似。分母为什么是$M-1$，这需要说明。$s_{i,j}$的期望是：

$$E_{s_{i,j}} = E\left(\frac{1}{M-1}\sum_{k=1}^{M}\left(X_i^k - \bar{X}_i\right)\left(X_j^k - \bar{X}_j\right)\right) = \frac{1}{M-1}\sum_{k=1}^{M}\left(E_{X_i^k X_j^k} - E_{X_i^k \bar{X}_j} - E_{X_j^k \bar{X}_i} + E_{\bar{X}_i \bar{X}_j}\right) \quad (5.27)$$

考察式（5.27）中的每一项。根据式（5.12），第一项可以变形：

$$E_{X_i^k X_j^k} = \mathrm{cov}_{X_i^k, X_j^k} + E_{X_i^k} E_{X_j^k} = \mathrm{cov}_{X_i, X_j} + E_{X_i} E_{X_j} \quad (5.28)$$

式（5.28）成立是因为X_i^k与X_i同分布，X_j^k与X_j同分布。式（5.27）的第二项是：

$$E_{X_i^k \bar{X}_j} = \mathrm{cov}_{X_i^k, \bar{X}_j} + E_{X_i^k} E_{\bar{X}_j} = \frac{1}{M}\sum_{s=1}^{M}\mathrm{cov}_{X_i^k, X_j^s} + E_{X_i} E_{X_j} = \frac{1}{M}\mathrm{cov}_{X_i, X_j} + E_{X_i} E_{X_j} \quad (5.29)$$

样本之间的独立性使$\mathrm{cov}_{X_i^k, X_j^s} = 0$（$k \neq s$），所以加和式的$M$项中只剩下$\mathrm{cov}_{X_i^k, X_j^k}$。$X_i^k$与$X_i$同分布，$X_j^k$与$X_j$同分布，所以$\mathrm{cov}_{X_i^k, X_j^k} = \mathrm{cov}_{X_i, X_j}$。另外$\bar{X}_j$和$X_j$的期望相同，所以式（5.29）成立。基于同样的证明，式（5.27）的第三项也等于同样的值。最后考察式（5.27）的第四项：

$$E_{\bar{X}_i \bar{X}_j} = \mathrm{cov}_{\bar{X}_i, \bar{X}_j} + E_{\bar{X}_i} E_{\bar{X}_j} = \frac{1}{M^2}\sum_{s=1}^{M}\sum_{t=1}^{M}\mathrm{cov}_{X_i^s, X_j^t} + E_{X_i} E_{X_j} = \frac{1}{M}\mathrm{cov}_{X_i, X_j} + E_{X_i} E_{X_j} \quad (5.30)$$

因为样本间的独立性，双重加和项中只有s与t相等的M个项不为 0，且都等于cov_{X_i,X_j}。另外，$\bar{X}_i$与X_i的期望相同，$\bar{X}_j$与X_j的期望相同，所以式（5.30）成立。将这四项代入式（5.27）得到：

$$E_{s_{i,j}} = \frac{1}{M-1}\sum_{k=1}^{M}\frac{M-1}{M}\mathrm{cov}_{X_i, X_j} = \mathrm{cov}_{X_i, X_j} \quad (5.31)$$

所以$s_{i,j}$是cov_{X_i,X_j}的无偏估计。为了无偏，分母要取$M-1$。以$s_{i,j}$为元素构成样本协方差矩阵$\boldsymbol{S}$，它是对协方差矩阵cov_X的无偏估计。根据式（5.26），可以把样本协方差矩阵$\boldsymbol{S}$写成：

$$\boldsymbol{S} = \frac{1}{M-1}(\boldsymbol{X} - \bar{\boldsymbol{X}})^{\mathrm{T}}(\boldsymbol{X} - \bar{\boldsymbol{X}}) \quad (5.32)$$

其中，$\boldsymbol{X}$是设计矩阵，其每一行是一个样本$\boldsymbol{x}^i$（$i = 1, \cdots, m$）。$\bar{\boldsymbol{X}}$是$M \times n$的矩阵，它的每一行都相同，都是样本均值$\bar{\boldsymbol{X}} = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}^i$的转置。式（5.32）容易验证，这里不再赘述。将样本“中心化”就是对每个分量减去该分量在样本集的均值，中心化之后的设计矩阵是：

$$\boldsymbol{X}' = \boldsymbol{X} - \overline{\boldsymbol{X}} \tag{5.33}$$

为了论述简洁，后文假设样本都已经中心化，并直接用$\boldsymbol{X}$而不是$\boldsymbol{X}'$表示已中心化的样本集。根据式（5.32），$\boldsymbol{S} = \frac{1}{M-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$是样本协方差矩阵。$\boldsymbol{S}$的对角线元素$s_{i,i}$是第$i$分量的样本方差，非对角线元素$s_{i,j}$是第$i$分量和第$j$分量的样本协方差。$\boldsymbol{S}$是对称矩阵，因为根据式（5.26），$s_{i,j}$与$s_{j,i}$相等。$\boldsymbol{S}$是半正定的，因为对于任意向量$\boldsymbol{y}$，有：

$$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} = \frac{1}{M-1}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{y} = \frac{1}{M-1}(\boldsymbol{X}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{X}\boldsymbol{y} = \frac{1}{M-1}\langle\boldsymbol{X}\boldsymbol{y}, \boldsymbol{X}\boldsymbol{y}\rangle \geqslant 0 \tag{5.34}$$

如果$\boldsymbol{X}$的列线性独立，则样本协方差矩阵$\boldsymbol{S}$正定，因为对于任意非零向量$\boldsymbol{y}$都有$\boldsymbol{X}\boldsymbol{y} \neq \boldsymbol{0}$，于是$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} = \frac{1}{M-1}\langle\boldsymbol{X}\boldsymbol{y}, \boldsymbol{X}\boldsymbol{y}\rangle > 0$，所以$\boldsymbol{S}$正定。正定矩阵的所有特征值为正，利用式（4.80）可构造$\boldsymbol{S}$的逆矩阵。

反之，如果$\boldsymbol{X}$的列线性相关，则样本协方差矩阵$\boldsymbol{S}$不可逆，因为存在非零向量$\boldsymbol{y}$使$\boldsymbol{X}\boldsymbol{y} = \boldsymbol{0}$，于是$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} = 0$，将$\boldsymbol{S}$谱分解为$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}$，令$\boldsymbol{z} = \boldsymbol{V}\boldsymbol{y}$，因为$\boldsymbol{V}$的列线性独立且$\boldsymbol{y} \neq \boldsymbol{0}$，所以$\boldsymbol{z} \neq \boldsymbol{0}$且：

$$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{y} = \boldsymbol{z}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{z} = \sum_{i=1}^{M}\lambda^{i}\,\mathbf{z}_{i}{}^{2} \tag{5.35}$$

因为$\boldsymbol{z}$必有非 0 分量，如果$\boldsymbol{S}$的特征值都大于 0，于是根据式（5.35）有$\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} > 0$，产生矛盾。所以$\boldsymbol{S}$必有 0 特征值。令 0 特征值对应的特征向量是$\boldsymbol{v}$，若$\boldsymbol{S}$可逆，则有：

$$\boldsymbol{v} = \boldsymbol{S}^{-1}\boldsymbol{S}\boldsymbol{v} = \boldsymbol{S}^{-1}(0 \times \boldsymbol{v}) = \boldsymbol{0} \tag{5.36}$$

这与特征向量不为零向量矛盾，所以$\boldsymbol{S}$必然不可逆。设计矩阵$\boldsymbol{X}$的列线性相关就是样本集的各个特征线性相关，这种情况称样本集存在多重共线性（multi-collinearity）。

5.1.5　主成分

样本协方差矩阵不一定是对角矩阵，即它的非对角线元素不一定为 0，也就是说各分量之间的样本协方差不一定为 0，它们有可能呈一定程度的线性关系，如图 5-2 所示。

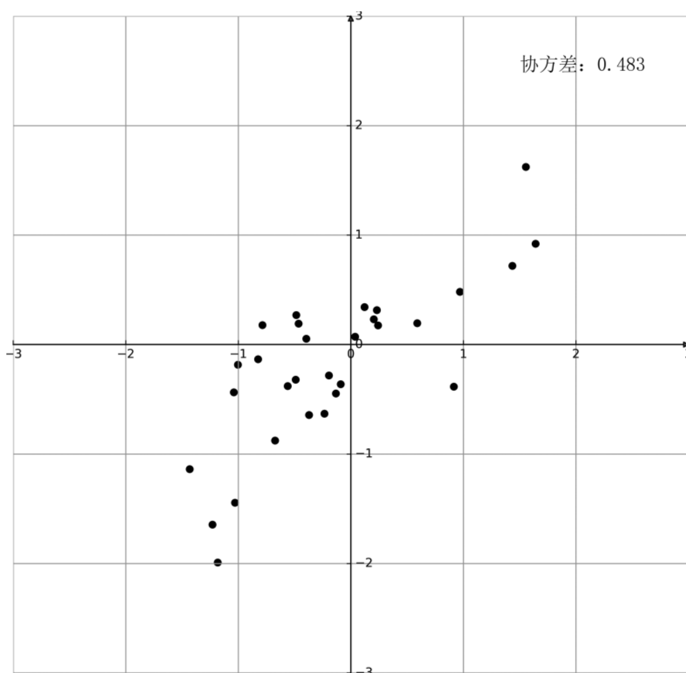


图 5-2　两个分量之间呈一定程度的线性关系

现在对每一个样本$\boldsymbol{x}^i$构造新的样本$\boldsymbol{y}^i = \boldsymbol{V}\boldsymbol{x}^i$，$\boldsymbol{V}$是样本协方差矩阵的谱分解中的正交矩阵，它的行是$\mathbb{R}^n$的一组标准正交基，$\boldsymbol{y}^i$的分量是样本$\boldsymbol{x}^i$向这组基的投影（坐标）。新样本的设计矩阵是：

$$\boldsymbol{Y} = \begin{pmatrix} (\boldsymbol{y}^1)^{\mathrm{T}} \\ \vdots \\ (\boldsymbol{y}^m)^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} (\boldsymbol{V}\boldsymbol{x}^1)^{\mathrm{T}} \\ \vdots \\ (\boldsymbol{V}\boldsymbol{x}^m)^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} (\boldsymbol{x}^1)^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}} \\ \vdots \\ (\boldsymbol{x}^m)^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}} \end{pmatrix} = \boldsymbol{X}\boldsymbol{V}^{\mathrm{T}} \tag{5.37}$$

因为样本集$\boldsymbol{x}^i$（$i = 1, \cdots, M$）经过中心化，即$\overline{\boldsymbol{x}} = \boldsymbol{0}$，所以新样本集的样本均值是：

$$\overline{\boldsymbol{y}} = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{V}\boldsymbol{x}^i = \boldsymbol{V}\left(\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}^i\right) = \boldsymbol{V}\overline{\boldsymbol{x}} = \boldsymbol{0} \tag{5.38}$$

所以新样本集也是中心化的。新样本集的样本协方差矩阵是：

$$\frac{1}{M-1}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y} = \frac{1}{M-1}\boldsymbol{V}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{\Lambda} \tag{5.39}$$

新样本集的样本协方差矩阵是对角阵，它的各分量不相关。新样本的第一个分量称为原样本的第一主成分（first principle component）。第一主成分的样本方差是$\boldsymbol{S}$的第一大特征值λ^1。λ^1的特征向量$\boldsymbol{v}^1$称为第一主成分方向。新样本的第二个分量称为第二主成分（second principle

component)。第二主成分的样本方差是$\boldsymbol{S}$的第二大特征值λ^2。λ^2的特征向量$\boldsymbol{v}^2$称为第二主成分方向，以此类推。

上述过程叫作主成分分析（principle component analysis）。主成分分析的本质是通过对样本协方差矩阵进行谱分解，得到一个新的坐标系。新坐标系的第一个坐标轴沿着第一主成分方向，样本在其上的投影称为第一主成分，第一主成分具有最大方差。第二个坐标轴沿着第二主成分方向，样本在其上的投影称为第二主成分，第二主成分具有第二大方差，以此类推。样本的各个主成分之间不相关。

主成分分析相当于旋转坐标轴，使样本集最分散的方向沿着第一坐标轴，垂直于第一坐标轴的第二分散的方向沿着第二坐标轴，以此类推，如图 5-3 所示。

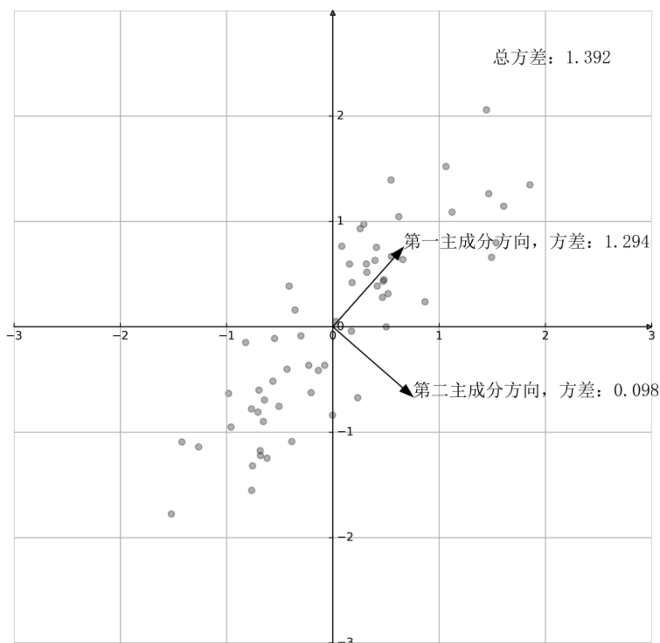


图 5-3　主成分分析相当于旋转坐标系

新样本协方差矩阵的迹等于原样本协方差矩阵的迹：

$$\operatorname{tr}\left(\frac{1}{M-1}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}\right)=\operatorname{tr}(\boldsymbol{V}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}})=\operatorname{tr}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{S})=\operatorname{tr}(\boldsymbol{S}) \tag{5.40}$$

这说明新样本各分量的样本方差之和等于原样本各分量的样本方差之和。也就是说，主成分保留了原样本集的变化信息，但去除了分量与分量之间的相关性。

主成分可用作数据降维。假设原样本为n维，但如果前$k \ll n$个主成分的方差之和占了总方差的绝大部分，例如 95%，则说明样本主要在前k个主成分方向上有差别，在其他主成分方向上差别很小，可视为噪声。抛弃后$n-k$个主成分，就在不损失有用信息的前提下，将数据从n维降到了k维，如图 5-4 所示。

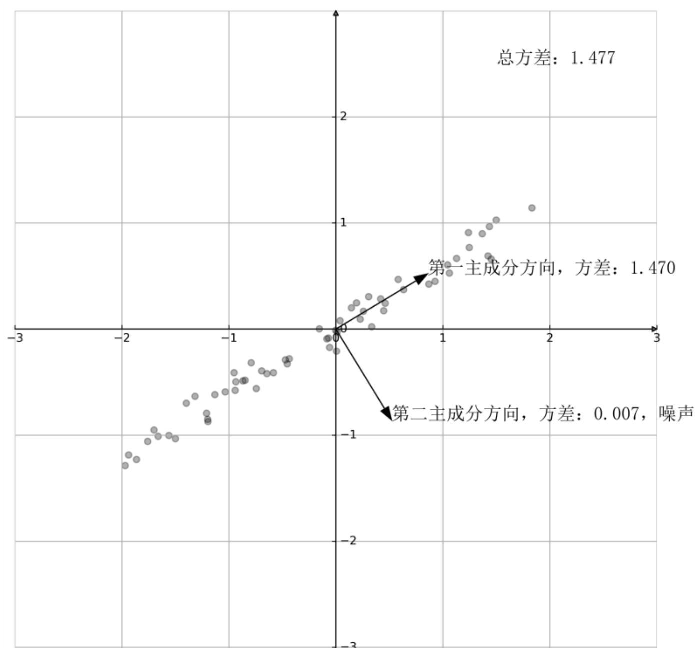


图 5-4 主成分用作降维，反映数据的真实维度

5.1.6 正态分布

本节讨论正态分布（normal distribution），让我们先从最简单的情况开始。若随机变量X的概率密度函数是：

$$p_X(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{x^2}{2}} \tag{5.41}$$

则称X服从正态分布$\mathcal{N}(0,1)$，记为$X \sim \mathcal{N}(0,1)$。首先需要证明式（5.41）满足概率密度函数的要求。$p_X(x) \geqslant 0$和连续性没有问题，需要证明$\int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x = 1$，我们看：

$$\left(\int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x\right)^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}x \cdot \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{y^2}{2}}\,\mathrm{d}y = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathrm{e}^{-\frac{x^2+y^2}{2}}\,\mathrm{d}x\,\mathrm{d}y \tag{5.42}$$

转化为在极坐标下积分，式（5.42）等于：

$$\left(\int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x\right)^2 = \frac{1}{2\pi}\int_0^{2\pi}\int_0^{\infty} r\mathrm{e}^{-\frac{r^2}{2}}\mathrm{d}r\,\mathrm{d}\theta = \frac{1}{2\pi}\cdot 2\pi = 1 \tag{5.43}$$

这就证明了$\int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x = 1$，$p_X(x)$确实是一个概率密度函数。$p_X(x)$是偶函数，则$xp_X(x)$是奇函数，奇函数在实数轴上积分为 0，所以$X$在$p_X$下的期望是 0。欲求$X$在$p_X$下的方差，利用分部积分公式$\int u\mathrm{d}v = uv - \int v\mathrm{d}u$，令$u = -x$，$v = \mathrm{e}^{-x^2/2}$，有：

$$\begin{aligned}\mathrm{var}_{p_X}(X) &= \int_{-\infty}^{\infty} x^2\,p_X(x)\,\mathrm{d}x = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} x^2\mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}x = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}(-x)\,\mathrm{d}\mathrm{e}^{-\frac{x^2}{2}} \\ &= -\frac{1}{\sqrt{2\pi}}x\mathrm{e}^{-\frac{x^2}{2}}\Big|_{-\infty}^{\infty} + \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}x = 1\end{aligned} \tag{5.44}$$

式（5.44）最后一个等号后有两项，根据洛必达法则，第一项在正负无穷处的极限都是 0，所以该项是 0，第二项就是p_X在实数轴上的积分，为 1，所以X的方差为 1。这就是$\mathcal{N}(0,1)$的含义：均值为 0，方差为 1。$\mathcal{N}(0,1)$称为标准正态分布，其概率密度如图 5-5 所示。

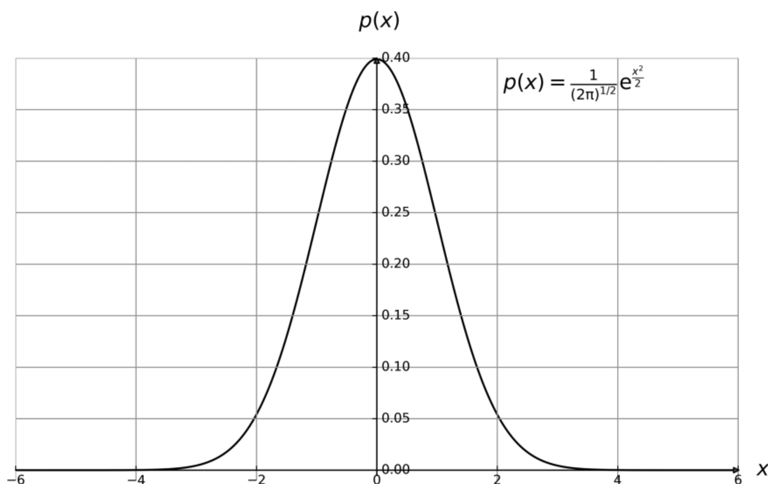


图 5-5　一元标准正态分布的概率密度函数

如果$X' \sim \mathcal{N}(0,1)$，考察随机变量$X = \sigma X' + \mu$。若X落在区间$(-\infty, x)$中，则X'落在区间$\left(-\infty, \frac{x-\mu}{\sigma}\right)$中，于是$X$落在$(-\infty, x)$中的概率是：

$$P(X < x) = P\left(X' < \frac{x-\mu}{\sigma}\right) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{x-\mu}{\sigma}}\mathrm{e}^{-\frac{u^2}{2}}\mathrm{d}u \tag{5.45}$$

令$v = \sigma u + \mu$，有$\mathrm{d}u = \frac{1}{\sigma}\mathrm{d}v$。对式（5.45）中的积分做变量变换：

$$P(X < x) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{x}\mathrm{e}^{-\frac{(v-\mu)^2}{2\sigma^2}}\mathrm{d}v \tag{5.46}$$

所以X的概率密度函数是：

$$p_X(x)=\frac{\mathrm{d}P(X<x)}{\mathrm{d}x}=\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.47}$$

因为$X=\sigma X'+\mu$，且$X'\sim\mathcal{N}(0,1)$，所以X的期望是：

$$E_X=\sigma E_{X'}+\mu=\mu \tag{5.48}$$

X的方差是：

$$\mathrm{var}_X=\sigma^2\mathrm{var}_{X'}=\sigma^2 \tag{5.49}$$

称X服从期望为μ，方差为σ^2的正态分布，记为$X\sim\mathcal{N}(\mu,\sigma^2)$。$X$的概率密度如图 5-6 所示。

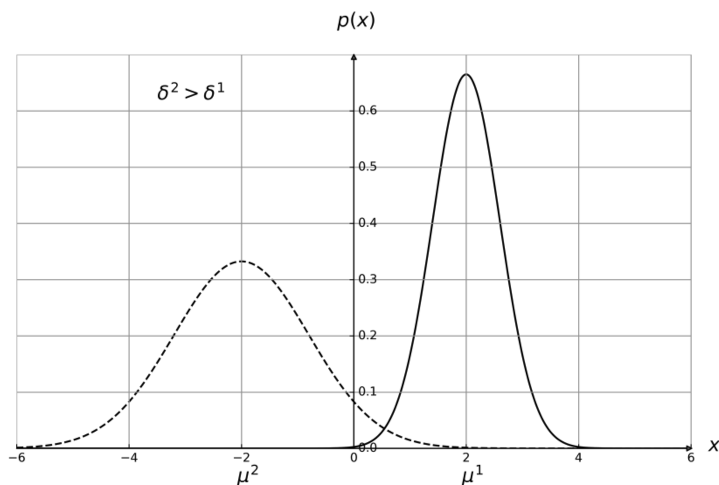


图 5-6　一元正态分布的概率密度函数（图中画出了两个）

现在我们从一元正态分布进入多元正态分布（multivariate normal distribution）。假设有n个独立的一元正态分布$X_i\sim\mathcal{N}(\mu_i,{\sigma_i}^2)$（$i=1,\cdots,n$）。以$X_i$（$i=1,\cdots,n$）为分量构成$n$维随机向量$X=(X_1\quad\cdots\quad X_n)^{\mathrm{T}}$。因为$X_i$（$i=1,\cdots,n$）独立，所以$X$的概率密度是$X_i$（$i=1,\cdots,n$）的概率密度的乘积：

$$p_X(\boldsymbol{x})=\prod_{i=1}^{n}p_{X^i}(x_i)=\prod_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma_i}\mathrm{e}^{-\frac{(x_i-\mu_i)^2}{2{\sigma_i}^2}}=\frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\mathrm{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}} \tag{5.50}$$

其中，$\boldsymbol{\mu}=(\mu_1\quad\cdots\quad\mu_n)^{\mathrm{T}}$是由$n$个期望组成的向量，$\boldsymbol{\Sigma}$是对角矩阵，对角线元素是$n$个方差$\sigma_i^2$（$i=1,\cdots,n$）。因为$X_i$（$i=1,\cdots,n$）独立，它们两两之间协方差为 0，所以对角阵$\boldsymbol{\Sigma}$就是随机向

量X的协方差矩阵。对角阵$\boldsymbol{\Sigma}$的行列式$|\boldsymbol{\Sigma}|$等于它的对角线元素之积，所以$|\boldsymbol{\Sigma}|^{1/2}$就等于$\prod_{i=1}^{n}\sigma_i$。称$X$服从期望为$\boldsymbol{\mu}$，协方差矩阵为$\boldsymbol{\Sigma}$的多元正态分布，记为$X\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$。分量之间不相关的二元正态分布的概率密度函数如图 5-7 所示。

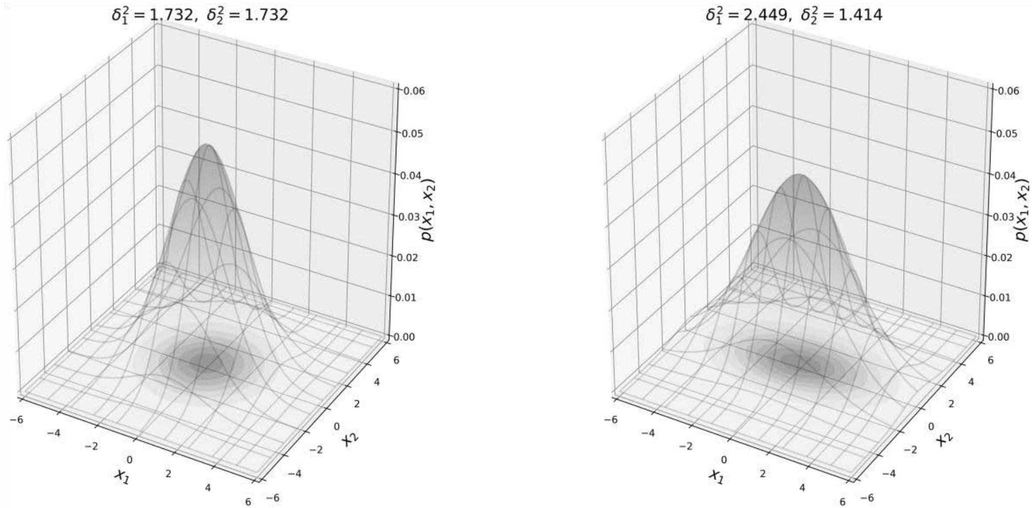


图 5-7　分量不相关的二元正态分布的概率密度函数

观察图 5-7 的图形，概率密度的峰值位置可以通过期望向量$\boldsymbol{\mu}$来调整，概率密度的等高线是椭圆，它的偏心率受控于各个分量的方差。现在我们旋转概率密度函数。只旋转而不拉伸的话，函数在$\mathbb{R}^n$上的积分保持不变，所以旋转之后仍是一个概率密度函数。将随机向量X向任意正交矩阵$\boldsymbol{V}$的行投影，得到新的随机向量$X'=\boldsymbol{V}X$。假设X'服从$\mathcal{N}(\boldsymbol{\mu}',\boldsymbol{\Sigma}')$，$\boldsymbol{\Sigma}'$是对角阵，也就是说$X$经过旋转后服从分量不相关的多元正态分布，则$X$的概率密度是：

$$p_X(\boldsymbol{x})=\frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}'|^{1/2}}\mathrm{e}^{-\frac{(\boldsymbol{Vx}-\boldsymbol{\mu}')^{\mathrm{T}}(\boldsymbol{\Sigma}')^{-1}(\boldsymbol{Vx}-\boldsymbol{\mu}')}{2}}=\frac{1}{(2\pi)^{n/2}|\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Sigma}'\boldsymbol{V}|^{1/2}}\mathrm{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\mu}')^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Sigma}')^{-1}\boldsymbol{V}(\boldsymbol{x}-\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\mu}')}{2}}$$
$$=\frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\mathrm{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}} \tag{5.51}$$

其中，$\boldsymbol{\mu}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\mu}'$，$\boldsymbol{\Sigma}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Sigma}'\boldsymbol{V}$。矩阵的行列式满足$|\boldsymbol{AB}|=|\boldsymbol{A}|\cdot|\boldsymbol{B}|$以及$|\boldsymbol{A}^{\mathrm{T}}|=|\boldsymbol{A}|$。因为$|\boldsymbol{V}|^2=|\boldsymbol{V}^{\mathrm{T}}|\cdot|\boldsymbol{V}|=|\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}|=|\boldsymbol{I}|=1$，所以$|\boldsymbol{\Sigma}|=|\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Sigma}'\boldsymbol{V}|=|\boldsymbol{V}|^2\cdot|\boldsymbol{\Sigma}'|=|\boldsymbol{\Sigma}'|$。又因为$\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Sigma}')^{-1}\boldsymbol{V}\boldsymbol{\Sigma}=\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Sigma}')^{-1}\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Sigma}'\boldsymbol{V}=\boldsymbol{I}$，所以$\boldsymbol{\Sigma}^{-1}=\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Sigma}')^{-1}\boldsymbol{V}$。$X$的期望是：

$$E_X=E_{\boldsymbol{V}^{\mathrm{T}}X'}=\boldsymbol{V}^{\mathrm{T}}E_{X'}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\mu}'=\boldsymbol{\mu} \tag{5.52}$$

X的协方差矩阵是：

$$\mathrm{cov}_X = \mathrm{cov}_{\boldsymbol{V}^{\mathrm{T}}X'} = \boldsymbol{V}^{\mathrm{T}}\mathrm{cov}_{X'}\boldsymbol{V} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Sigma}'\boldsymbol{V} = \boldsymbol{\Sigma} \tag{5.53}$$

$\boldsymbol{\Sigma}$不一定是对角阵，所以X的各分量之间可能相关，如图 5-8 所示。

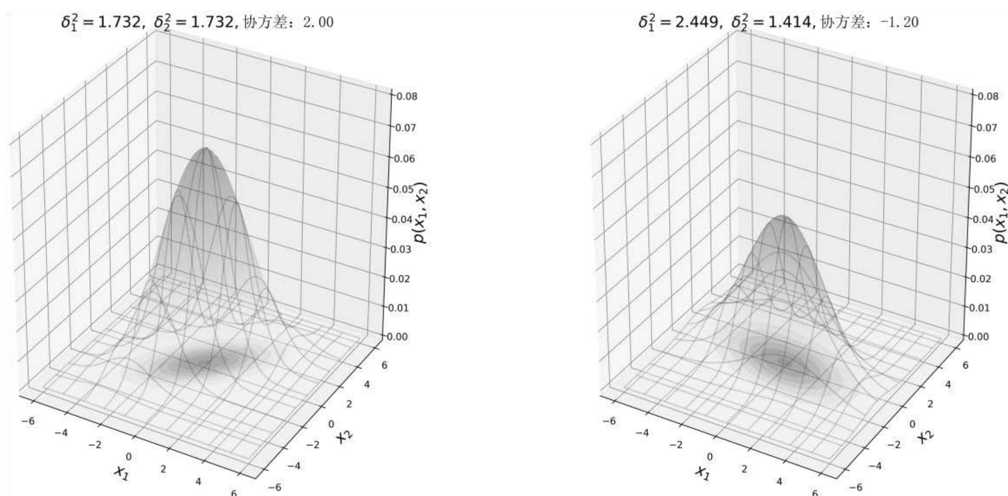


图 5-8　二元正态分布的概率密度函数

我们称X服从期望是$\boldsymbol{\mu}$，协方差矩阵是$\boldsymbol{\Sigma}$的多元正态分布，记为$X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$。$X$的概率密度函数是：

$$p_X(\boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \mathrm{e}^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}} \tag{5.54}$$

基于与（5.51）同样的证明，如果$X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$且$X' = \boldsymbol{V}\boldsymbol{X} + \boldsymbol{\mu}'$，$\boldsymbol{V}$是正交矩阵，则$X' \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\mu}', \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}})$。

5.2　模型自由度与偏置–方差权衡

模型的自由度与模型参数的数量有关，也与训练过程中对参数变化的约束有关。例如，逻辑回归模型有n个权值和 1 个偏置。如果允许在$\mathbb{R}^{n+1}$空间中自由地搜索最优的$\boldsymbol{w}$和b，则逻辑回归模型的自由度就是$n+1$，但如果参数的搜索是受限的，则自由度就可能小于$n+1$。

自由度控制模型的偏置–方差权衡。模型的预测误差可以分解为数据本身噪声的方差、模型预测的偏置的平方（注意，这个偏置不是指线性模型的b，后文会详解）以及模型预测的方差三项，后两项此消彼长。正则化控制模型自由度，是调整偏置–方差权衡的重要手段。弱正则化导致高自由度，模型处于低偏置–高方差状态；强正则化导致低自由度，模型处于高偏置–低方差状态。

5.2.1　最小二乘线性回归

线性回归模型的计算式是：

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{5.55}$$

式（5.55）非常简单，它是输入向量$\boldsymbol{x}$的仿射函数。可以将其视为没有 Logistic 函数的逻辑回归，也可以视为没有激活函数，或激活函数是恒等函数的神经元，如图 5-9 所示。

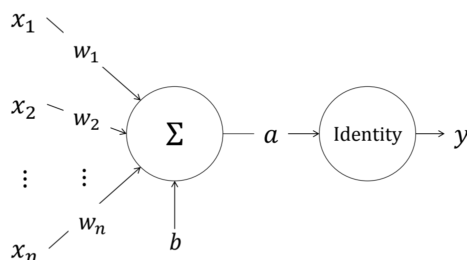


图 5-9　将线性回归视为单个神经元

与逻辑回归不同，线性回归的输出不限于$(0,1)$区间，不能视为概率。线性回归的输出是对真实值的预测。线性回归的训练集形如$\{\boldsymbol{x}^i, y^i\}_{i=1}^M$，其中$\boldsymbol{x}^i$（$i = 1,\cdots,M$）都是$n$维向量，$y^i$（$i = 1,\cdots,M$）是标量目标值。线性回归的目的是寻找最优的$\boldsymbol{w}$和$b$，对新样本预测其目标值。先用上一章使用过的技巧简化公式，为输入添加一维常量 1，使输入向量成为：

$$\boldsymbol{x} = \begin{pmatrix} x_0 = 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{5.56}$$

将偏置b纳入权值向量：

$$\boldsymbol{w} = \begin{pmatrix} w_0 = b \\ w_1 \\ \vdots \\ w_n \end{pmatrix} \tag{5.57}$$

线性回归的计算式就简化为$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$。以$M$个增加了一维的样本为行，构成$M \times (n+1)$的设计矩阵$\boldsymbol{X}$。以$M$个目标值构成$M$维向量$\boldsymbol{y}$。最小二乘线性回归，是指用最小二乘法训练线性回归。最小二乘法的损失函数是均方误差（mean-square error，MSE）：

$$\mathrm{loss}(\boldsymbol{w}) = \frac{1}{M}\sum_{i=1}^{M}(y^i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i)^2 \tag{5.58}$$

均方误差是训练集上的目标值与模型预测值误差平方的平均。可以用设计矩阵$\boldsymbol{X}$和目标值向量$\boldsymbol{y}$来表示均方误差：

$$\text{loss}(\boldsymbol{w}) = \frac{1}{M}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 = \frac{1}{M}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 \tag{5.59}$$

$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$是由预测值组成的$M$维向量。式（5.59）表明：均方误差正比于$\boldsymbol{y}$与$\hat{\boldsymbol{y}}$之间距离的平方，最小化均方误差就是最小化$\hat{\boldsymbol{y}}$与$\boldsymbol{y}$之间的距离。向量$\boldsymbol{X}\boldsymbol{w}$是$\boldsymbol{X}$的列的线性组合，属于$\boldsymbol{X}$的列空间。最小化均方误差就是在$\boldsymbol{X}$的列空间中寻找与$\boldsymbol{y}$距离最小的向量，即$\boldsymbol{y}$向$\boldsymbol{X}$的列空间的投影。此投影的系数$\boldsymbol{w}^*$称为线性回归的最小二乘解。

从另一个角度解释均方误差的意义，如果真实的权值是$\boldsymbol{w}$，则所有训练样本的真实输出应该是$\boldsymbol{X}\boldsymbol{w}$，但由于噪声，观察到的训练集目标值是：

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{e} \tag{5.60}$$

假设误差向量$\boldsymbol{e}$服从$\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Lambda})$，$\boldsymbol{0}$是$M$维零向量，$\boldsymbol{\Lambda}$是$M \times M$对角阵，对角线元素是$\sigma^2$。也就是说，每个样本的误差是独立同分布的，它们都服从$\mathcal{N}(0, \sigma^2)$。于是随机向量$\boldsymbol{y}$服从$\mathcal{N}(\boldsymbol{X}\boldsymbol{w}, \boldsymbol{\Lambda})$。真实权值向量为$\boldsymbol{w}$的前提下，$\boldsymbol{y}$的概率密度是：

$$p(\boldsymbol{y}|\boldsymbol{w}) = \frac{1}{(2\pi\sigma^2)^{M/2}} \mathrm{e}^{-\frac{(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})}{2\sigma^2}} \tag{5.61}$$

$p(\boldsymbol{y}|\boldsymbol{w})$是权值为$\boldsymbol{w}$的前提下，观察到目标值向量$\boldsymbol{y}$的似然概率。回顾式（5.59），使$\text{loss}(\boldsymbol{w})$最小的$\boldsymbol{w}^*$也使$p(\boldsymbol{y}|\boldsymbol{w})$最大，所以最小二乘解$\boldsymbol{w}^*$也是最大似然解。可以用梯度下降法或二阶算法优化均方误差来寻找最小二乘解$\boldsymbol{w}^*$，但我们通过令梯度为零向量来求此问题的解析解。

$$\nabla\text{loss}(\boldsymbol{w}) = \frac{2}{M}\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \tag{5.62}$$

将$\nabla\text{loss}(\boldsymbol{w})$置为零向量，解得最小二乘解是：

$$\boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{5.63}$$

式（5.63）的前提是$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$可逆，这要求$\boldsymbol{X}$的列线性独立，即训练集不存在多重共线性。如果线性独立，则$\boldsymbol{X}$的列是它的列空间的一组基。得到最小二乘解$\boldsymbol{w}^*$后，模型对训练样本的预测是：

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{P}\boldsymbol{y} \tag{5.64}$$

$\boldsymbol{P} = \boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}$称为投影矩阵（projection matrix）或帽子矩阵（hat matrix），因为它给训练集的目标向量$\boldsymbol{y}$带上“帽子”——$\hat{\boldsymbol{y}}$。

5.2.2　模型自由度

在线性回归的框架下可以给出模型自由度（degree of freedom）的精确定义——投影矩阵的秩。$\boldsymbol{w}^*$是使均方误差最小的权值，所以$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^*$是$\boldsymbol{X}$的列空间中与$\boldsymbol{y}$距离最近的点。观察$\boldsymbol{y}$与$\hat{\boldsymbol{y}}$的差：

$$\begin{aligned}\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y}-\hat{\boldsymbol{y}}) &= \boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}) = (\boldsymbol{X}^{\mathrm{T}}-\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}})\boldsymbol{y} \\ &= (\boldsymbol{X}^{\mathrm{T}}-\boldsymbol{X}^{\mathrm{T}})\boldsymbol{y} = \boldsymbol{0}\end{aligned} \tag{5.65}$$

其中，$\boldsymbol{y}-\hat{\boldsymbol{y}}$与$\boldsymbol{X}$的每一列正交，于是它正交于$\boldsymbol{X}$的列空间，所以$\hat{\boldsymbol{y}}$是$\boldsymbol{y}$向$\boldsymbol{X}$的列空间的投影。还有另一种方法看出这一点，将$\boldsymbol{X}$奇异值分解为$\boldsymbol{U\Gamma V}$。因为$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$可逆，所以$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$是正定的。于是$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$的$n+1$个特征值都大于 0，即$\boldsymbol{X}$有$n+1$个奇异值。奇异值分解中的$\boldsymbol{U}$是$M\times(n+1)$矩阵。$\boldsymbol{U}$的列是两两正交的标准向量，有$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{I}$。$\boldsymbol{\Gamma}$是$n+1$的对角阵，对角线元素是$\boldsymbol{X}$的奇异值$\sqrt{\lambda^1},\sqrt{\lambda^2},\cdots,\sqrt{\lambda^{n+1}}$。$\boldsymbol{V}$是正交矩阵。用奇异值分解替换$\boldsymbol{X}$，将式（5.64）展开：

$$\begin{aligned}\hat{\boldsymbol{y}} &= \boldsymbol{U\Gamma V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U\Gamma V})^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{U\Gamma V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^2\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y} \\ &= \boldsymbol{U\Gamma V V}^{\mathrm{T}}\boldsymbol{\Gamma}^{-2}\boldsymbol{V V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{U\Gamma\Gamma}^{-2}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}\end{aligned} \tag{5.66}$$

式（5.66）利用了$\boldsymbol{V}^{-1}=\boldsymbol{V}^{\mathrm{T}}$，$\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{I}$，$(\boldsymbol{AB})^{-1}=\boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$以及$\boldsymbol{\Gamma}^{\mathrm{T}}=\boldsymbol{\Gamma}$。后文还有不少利用奇异值分解的计算，都与该式类似，届时会简化推导过程。式（5.66）表明：$\hat{\boldsymbol{y}}$等于先求$\boldsymbol{y}$在列空间标准正交基上的坐标，再用该坐标将标准正交基线性组合。所以$\hat{\boldsymbol{y}}$是$\boldsymbol{y}$向$\boldsymbol{X}$的列空间的投影，如图 5-10 所示。

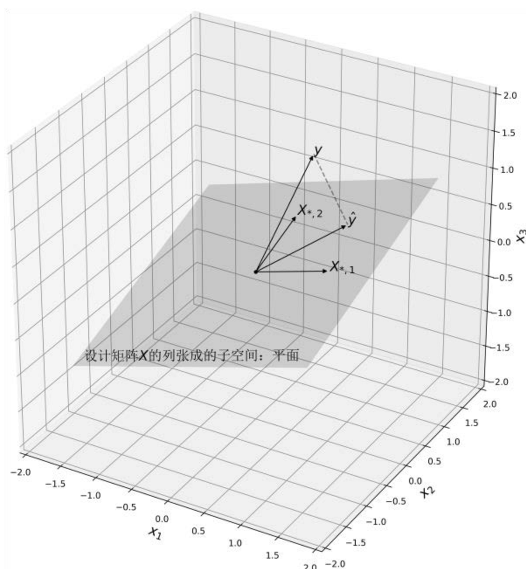


图 5-10　最小二乘解对训练集的预测值是目标值在设计矩阵列空间的投影

从这个角度可以看出，最小二乘线性回归完全自由地在列空间中寻找$\boldsymbol{y}$的投影。列空间的维数为$n+1$，最小二乘解是在$n+1$个方向上不受约束地搜索。我们还能看到，投影矩阵的迹是：

$$\mathrm{tr}(\boldsymbol{P})=\mathrm{tr}(\boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}})=\mathrm{tr}(\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}})=\mathrm{tr}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U})=\mathrm{tr}(\boldsymbol{I}_{n+1})=n+1 \tag{5.67}$$

投影矩阵的迹恰是$n+1$，所以将最小二乘线性回归的自由度定义为投影矩阵的迹。如果去掉一个样本分量（去掉一个特征），那么设计矩阵的形状将变成$M\times n$，设计矩阵的列空间的维数也就变成了n，投影矩阵的迹也成了n。所以去掉一个特征后，最小二乘线性回归的自由度就减小了 1，从$n+1$变成了n。

如果不是增减特征，而是给搜索施加某种约束，自由度该如何变化呢？约束的强度应该是连续的，它对自由度的影响不应是跳跃地增加或减小 1，而应该是连续的变化。这能否反映在投影矩阵的迹中呢？后文讲解正则化，并将正则化应用于线性回归后，这些问题就都能得到解释。

本节在线性回归框架下讨论模型自由度，并用投影矩阵的迹定义模型自由度。在最小二乘线性回归的情况下，投影矩阵的迹等于最小二乘解的自由搜索方向。对于更复杂的模型，例如神经网络、决策树等，自由度没有精确的定义。最小二乘线性回归的自由度等于模型参数个数。参数个数与模型自由度有关，参数越多则自由度越高。神经网络和深度学习的参数远远多于线性回归，所以它们也具有大得多的自由度。除了参数个数这个因素，模型自由度也与训练过程所受到的约束等其他因素有关。

5.2.3 偏置–方差权衡

和自由度的概念一样，我们仍在线性回归框架下讨论偏置–方差权衡。我们关心模型对新样本$\boldsymbol{x}_{\text{new}}$的预测误差，将训练样本$\boldsymbol{x}^i$（$i=1,\cdots,M$）视为固定，将训练目标值$y^i$（$i=1,\cdots,M$）视为随机，则最小二乘解$\boldsymbol{w}^*=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$是随机向量。若真实的权值向量是$\boldsymbol{w}$，假设$M$个训练目标值是：

$$y^i=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i+\epsilon^i,\quad \epsilon^i\sim\mathcal{N}(0,\sigma^2),\quad i=1,\cdots,M \tag{5.68}$$

其中ϵ^i（$i=1,\cdots,M$）是独立同分布的。因为存在噪声，新样本$\boldsymbol{x}_{\text{new}}$的观察值$y_{\text{new}}$也是随机变量，服从$\mathcal{N}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_{\text{new}},\sigma^2)$。模型对新样本的预测值是$\hat{y}_{\text{new}}=(\boldsymbol{w}^*)^{\mathrm{T}}\boldsymbol{x}_{\text{new}}$。$y_{\text{new}}$的随机性来源于噪声正态误差，$\hat{y}_{\text{new}}$的随机性来源于训练集目标值的噪声正态误差。这些正态误差都是独立的，所以y_{new}与$\hat{y}_{\text{new}}$独立。根据式（5.13），误差平方的期望是：

$$E((y_{\text{new}}-\hat{y}_{\text{new}})^2)=\mathrm{var}(y_{\text{new}}-\hat{y}_{\text{new}})+\big(E(y_{\text{new}}-\hat{y}_{\text{new}})\big)^2 \tag{5.69}$$

因为y_{new}与$\hat{y}_{\text{new}}$独立，它们的协方差为 0，有：

$$\text{var}(y_{\text{new}}-\hat{y}_{\text{new}})=\text{var}_{y_{\text{new}}}-2\text{cov}_{y_{\text{new}},\hat{y}_{\text{new}}}+\text{var}_{\hat{y}_{\text{new}}}=\text{var}_{y_{\text{new}}}+\text{var}_{\hat{y}_{\text{new}}} \quad (5.70)$$

所以误差平方的期望可以拆解为：

$$E((y_{\text{new}}-\hat{y}_{\text{new}})^2)=\text{var}_{y_{\text{new}}}+\text{var}_{\hat{y}_{\text{new}}}+\left(E_{y_{\text{new}}}-E_{\hat{y}_{\text{new}}}\right)^2 \quad (5.71)$$

其中，$E_{y_{\text{new}}}-E_{\hat{y}_{\text{new}}}$是新样本的预测值与观察值的期望之差，称为偏置（bias）。观察值的期望是真实值$\boldsymbol{w}^{\text{T}}\boldsymbol{x}_{\text{new}}$。式（5.71）的含义是误差平方的期望可以分解为三部分之和：观察值的方差、预测值的方差以及偏置的平方。式（5.71）不仅适用于线性回归，也适用于任何回归模型。

$\text{var}_{y_{\text{new}}}$来自问题本身，无法避免，$\text{var}_{\hat{y}_{\text{new}}}$和偏置产生于模型。最小二乘线性回归对新样本的预测值的期望是：

$$\begin{aligned}E_{\hat{y}_{\text{new}}}&=E((\boldsymbol{x}_{\text{new}})^{\text{T}}\boldsymbol{w}^*)=E((\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}\boldsymbol{y})=(\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}E(\boldsymbol{y})\\&=(\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}\boldsymbol{X}\boldsymbol{w}=(\boldsymbol{x}_{\text{new}})^{\text{T}}\boldsymbol{w}=E_{y_{\text{new}}}\end{aligned} \quad (5.72)$$

$E_{y_{\text{new}}}=E_{\hat{y}_{\text{new}}}$，所以最小二乘线性回归的偏置为 0，它是无偏模型，误差全部来自于预测值方差。最小二乘线性回归的预测值方差是：

$$\begin{aligned}\text{var}_{\hat{y}_{\text{new}}}&=\text{var}((\boldsymbol{x}_{\text{new}})^{\text{T}}\boldsymbol{w}^*)=\text{var}((\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}\boldsymbol{y})\\&=(\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}(\sigma^2\boldsymbol{I})\boldsymbol{X}((\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1})^{\text{T}}\boldsymbol{x}_{\text{new}}=\sigma^2(\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}}\end{aligned} \quad (5.73)$$

利用$\boldsymbol{X}^{\text{T}}\boldsymbol{X}$的谱分解可求得$(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}$，并将其代入式（5.73）：

$$\begin{aligned}\text{var}_{\hat{y}_{\text{new}}}&=\sigma^2(\boldsymbol{x}_{\text{new}})^{\text{T}}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}}=\sigma^2(\boldsymbol{V}\boldsymbol{x}_{\text{new}})^{\text{T}}\begin{pmatrix}\frac{1}{\lambda^1}&\cdots&0\\\vdots&\ddots&\vdots\\0&\cdots&\frac{1}{\lambda^{n+1}}\end{pmatrix}\boldsymbol{V}\boldsymbol{x}_{\text{new}}\\&=\sigma^2\cdot\left\|\boldsymbol{\Lambda}^{-1/2}\boldsymbol{V}\boldsymbol{x}_{\text{new}}\right\|^2\end{aligned} \quad (5.74)$$

其中，$\boldsymbol{\Lambda}^{-1/2}$是对角阵，对角线元素是$\frac{1}{\sqrt{\lambda^i}}$（$i=1,\cdots,n+1$）。如果训练集的各个特征都已经中心化，那么$\boldsymbol{X}^{\text{T}}\boldsymbol{X}$去掉第一行和第一列后，右下角的$n\times n$子方阵就是训练集的样本协方差矩阵（差一个常数因子$\frac{1}{M+1}$）。根据式（5.74），模型对新样本$\boldsymbol{x}_{\text{new}}$的预测值方差等于将$\boldsymbol{x}_{\text{new}}$向训练集样本协方差矩阵的各个主成分方向投影，然后对各个分量施加惩罚，惩罚因子是各主成分的标准差的倒数$\frac{1}{\sqrt{\lambda^i}}$（$i=1,\cdots,n+1$）。惩罚后的主成分向量的模乘上问题固有方差$\sigma^2$，就得到了预测值方差。

如果新样本$\boldsymbol{x}_{\text{new}}$比较符合训练样本的分布，将它投影到各个主成分方向后，它的各分量会受到适当的惩罚，从而预测值方差较小。但如果$\boldsymbol{x}_{\text{new}}$不太符合训练样本分布，是一个“离群点”，那么投影后它的各分量的相对大小与各奇异值的相对大小不符合，对各分量的惩罚将会不恰当，导致预测值方差较大，如图 5-11 所示。

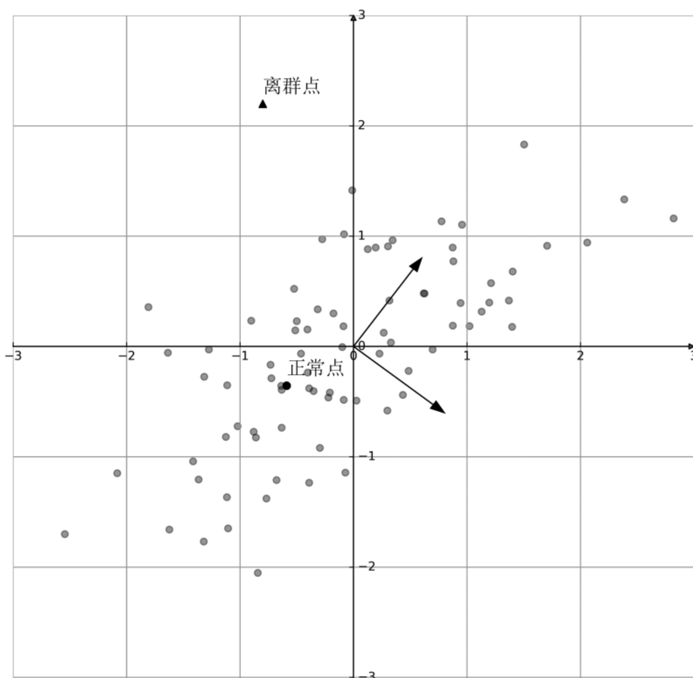


图 5-11　离群点在各主成分方向上的投影与各主成分的方差不匹配

后文会看到，强正则化降低模型自由度，减小预测值方差而增大偏置。这种现象不仅适用于线性回归，也适用于所有分类和回归模型。自由度控制着偏置–方差权衡：如果提高模型自由度，则偏置减小而方差增大；如果降低模型自由度，则偏置增大而方差减小。

低偏置并非建模追求的唯一目标，因为虽然低偏置时预测值的期望接近真实值，但如果方差过大，每一次预测的质量不可控制。反之，一定程度牺牲无偏性而减小方差，虽然预测值的期望有偏，但是方差小，多次预测整体的质量可能会更好，如图 5-12 所示。

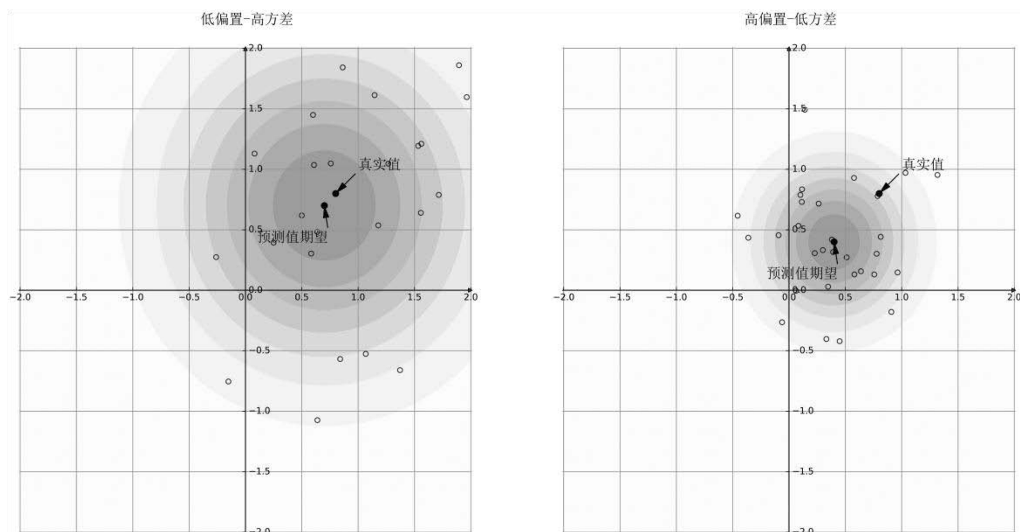


图 5-12　偏置与方差对预测的影响

更何况一般建模场景中对真实分布一无所知，这时选择某种模型（例如线性回归）只是出于对数据真实分布的先验的假设。过分追求低偏置，迫使模型无偏于假设的真实值期望，有可能导致过拟合。后文会讲解过拟合与欠拟合。

5.3　正则化

本节在线性回归框架下介绍$\mathcal{L}_2$正则化（$\mathcal{L}_2$ regularization），并从训练集样本协方差矩阵的主成分方向和贝叶斯先验概率两种视角阐述$\mathcal{L}_2$正则化的含义。之后我们简单介绍$\mathcal{L}_1$正则化，并比较$\mathcal{L}_1$和$\mathcal{L}_2$正则化的异同以及差异的原因。

5.3.1　岭回归与$\mathcal{L}_2$正则化

求线性回归最小二乘解$\boldsymbol{w}^*$要求$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$可逆，当$\boldsymbol{X}$的列线性相关时，这种可逆性不满足，但我们可以修正$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$：

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I} \tag{5.75}$$

将$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$谱分解，有：

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}+\lambda\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})\boldsymbol{V}=\boldsymbol{V}^{\mathrm{T}}\begin{pmatrix}\lambda^1+\lambda & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \lambda^{n+1}+\lambda\end{pmatrix}\boldsymbol{V} \tag{5.76}$$

根据式（5.76），有$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})\boldsymbol{V}^{\mathrm{T}}=\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})$，所以$\lambda^i+\lambda$（$i=1,\cdots,n+1$）是$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})$的$n+1$个（可重）特征值，正交矩阵$\boldsymbol{V}^{\mathrm{T}}$的列是对应的特征向量。只要$\lambda$足够大，就可以使$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}$的特征值都大于 0，从而可逆。用$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}$替换$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$，代入式（5.63），得到：

$$\boldsymbol{w}^{\mathrm{ridge}}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{5.77}$$

以$\boldsymbol{w}^{\mathrm{ridge}}$为线性回归的解，称作岭回归（ridge regression）。“岭”是指矩阵$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}$的对角线——山脊。我们知道最小二乘解$\boldsymbol{w}^*$最小化均方误差，那么岭回归的解$\boldsymbol{w}^{\mathrm{ridge}}$是否也是某个损失函数的最优解呢？请看损失函数：

$$\mathrm{loss}^{\mathrm{ridge}}(\boldsymbol{w})=\frac{1}{M}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^2+\frac{\lambda}{M}\|\boldsymbol{w}\|^2 \tag{5.78}$$

其中$\lambda>0$，称为正则化强度（regularization strength）。式（5.78）为均方误差加上了一项$\frac{\lambda}{M}\|\boldsymbol{w}\|^2$，它称为$\mathcal{L}_2$正则项，它是权值向量$\boldsymbol{w}$的长度平方的常数倍。

加入$\mathcal{L}_2$正则项后，损失函数除了惩罚预测误差还惩罚权值向量的长度，最优解需要兼顾这两种惩罚。可以想象，优化算法将无法在$\boldsymbol{X}$的列空间中自由地寻找$\boldsymbol{y}$的投影。这种约束就像一个“力”，将优化算法的解拉向原点。“力”的大小受控于λ和解与原点距离的平方，λ越大，则$\mathcal{L}_2$正则项在损失函数中占的比重越大，将解拉向原点的“力”越强。损失函数（5.78）的梯度是：

$$\nabla\mathrm{loss}^{\mathrm{ridge}}(\boldsymbol{w})=\frac{2}{M}\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})+\frac{2\lambda}{M}\boldsymbol{w}=\frac{2}{M}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})\boldsymbol{w}-\frac{2}{M}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{5.79}$$

令梯度为零向量，求得的解正是$\boldsymbol{w}^{\mathrm{ridge}}$。所以岭回归的解就是带$\mathcal{L}_2$正则项的均方误差损失函数的最优解。

虽然增高$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$的“岭”是为了克服训练集的多重共线性，但$\mathcal{L}_2$正则化已经不仅仅是为了这个目的，$\mathcal{L}_2$正则化是对自由度的控制。下面考察正则化对自由度的影响，为了与最小二乘线性回归比较，以下假设$\boldsymbol{X}$的列线性独立，即$\boldsymbol{X}$有$n+1$个奇异值。岭回归的投影矩阵是：

$$\boldsymbol{P}^{\mathrm{ridge}}=\boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}} \tag{5.80}$$

利用$\boldsymbol{X}$的奇异值分解$\boldsymbol{X}=\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}$，将$\boldsymbol{P}^{\mathrm{ridge}}$写为：

$$\begin{aligned}\boldsymbol{P}^{\mathrm{ridge}}&=\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{V}+\lambda\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{U}\boldsymbol{\Gamma}(\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\Gamma}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\\&=\boldsymbol{U}\begin{pmatrix}\frac{\lambda^1}{\lambda^1+\lambda}&\cdots&0\\\vdots&\ddots&\vdots\\0&\cdots&\frac{\lambda^{n+1}}{\lambda^{n+1}+\lambda}\end{pmatrix}\boldsymbol{U}^{\mathrm{T}}\end{aligned} \tag{5.81}$$

式（5.81）表明：岭回归的投影矩阵将目标值向量$\boldsymbol{y}$向设计矩阵列空间的标准正交基投影，之后压缩每个分量，所以岭回归的预测值向量偏向原点，而不再是目标值的垂直投影。根据自由度的定义，岭回归的自由度是投影矩阵的迹，因为$\mathrm{tr}(\boldsymbol{AB})=\mathrm{tr}(\boldsymbol{BA})$，于是有：

$$\mathrm{tr}\left(\boldsymbol{P}^{\mathrm{ridge}}\right)=\mathrm{tr}\left(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}\begin{pmatrix}\frac{\lambda^1}{\lambda^1+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\lambda^{n+1}}{\lambda^{n+1}+\lambda}\end{pmatrix}\right)=\sum_{i=1}^{n+1}\frac{\lambda^i}{\lambda^i+\lambda}<n+1 \tag{5.82}$$

$\mathcal{L}_2$正则项使岭回归的自由度小于$n+1$。正则化强度λ控制模型自由度，$\lambda=0$时就是最小二乘线性回归，自由度为$n+1$，λ趋近于正无穷则自由度趋近于 0。设想若真有无穷大的正则化强度，则$\boldsymbol{w}^{\mathrm{ridge}}$只能是零向量，否则一旦它有非 0 的长度，则损失函数马上变成正无穷。

以投影矩阵的迹为自由度无非是一个定义，重要的是它的含义。就像电影“黑客帝国”中印度程序人所说：Love is a word. What matters is the connection the word implies.（爱只是一个词，重要的是它的含义），我们来看$\mathcal{L}_2$正则化对偏置和方差的影响。岭回归对新样本$\boldsymbol{x}_{\mathrm{new}}$的预测值是：

$$\hat{y}_{\mathrm{new}}=(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{5.83}$$

如果真实权值向量是$\boldsymbol{w}$，则$\hat{y}_{\mathrm{new}}$的期望是：

$$E_{\hat{y}_{\mathrm{new}}}=(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}E_{\boldsymbol{y}}=(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} \tag{5.84}$$

新样本观察值y_{new}的期望是$(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}\boldsymbol{w}$。再次利用奇异值分解：

$$\begin{aligned}E_{y_{\mathrm{new}}}-E_{\hat{y}_{\mathrm{new}}}&=(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}\boldsymbol{w}-(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\Gamma}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\Gamma}\boldsymbol{V}\boldsymbol{w}\\&=(\boldsymbol{V}\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}\begin{pmatrix}\frac{\lambda}{\lambda^1+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\lambda}{\lambda^{n+1}+\lambda}\end{pmatrix}\boldsymbol{V}\boldsymbol{w}\end{aligned} \tag{5.85}$$

λ为 0 时，$E_{y_{\mathrm{new}}}-E_{\hat{y}_{\mathrm{new}}}=0$，模型无偏，$\lambda$越大则偏置越大，即强正则化降低模型自由度而增大偏置。接下来再看方差，利用谱分解求得$\hat{y}_{\mathrm{new}}$的方差是：

$$\begin{aligned}\mathrm{var}_{\hat{y}_{\mathrm{new}}}&=\sigma^2(\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{x}_{\mathrm{new}}\\&=\sigma^2(\boldsymbol{V}\boldsymbol{x}_{\mathrm{new}})^{\mathrm{T}}\begin{pmatrix}\frac{\lambda^1}{(\lambda^1+\lambda)^2} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\lambda^{n+1}}{(\lambda^{n+1}+\lambda)^2}\end{pmatrix}\boldsymbol{V}\boldsymbol{x}_{\mathrm{new}}\end{aligned} \tag{5.86}$$

λ为 0 时，岭回归退化成最小二乘线性回归，随着λ增大，式（5.86）中对角阵的对角线元素减小，导致$\mathrm{var}_{\hat{y}_{\mathrm{new}}}$减小。当$\lambda$趋近于正无穷$\mathrm{var}_{\hat{y}_{\mathrm{new}}}$趋近于 0。

$\boldsymbol{V}$的行是训练集样本协方差矩阵的主成分方向，样本在第一个主成分方向上的分量有最大的样本方差λ^1，在第二个主成分方向上的分量有次大的样本方差λ^2，以此类推。式（5.86）将$\boldsymbol{x}_{\mathrm{new}}$向各主成分方向投影，然后对每个投影分量施以惩罚。惩罚因子分母上的λ使方差更小，大方差方向受到的惩罚相对较小，小方差方向受到的惩罚相对较大。方差小的方向可以视作噪声，强正则化加重了对噪声的惩罚。

小结一下，当采用均方误差作为损失函数时，线性回归算法称为最小二乘线性回归。给均方误差加上$\mathcal{L}_2$正则项就成为岭回归。最小二乘线性回归的自由度是特征数加 1，它的预测无偏，但方差较大。$\mathcal{L}_2$正则化越强，则岭回归的自由度越低，预测的偏置较大，但方差较小。正则化强度控制模型自由度，进行偏置–方差权衡。

5

5.3.2 $\mathcal{L}_2$正则化的贝叶斯视角

权值向量为$\boldsymbol{w}$时，训练集的目标值服从式（5.68）的独立正态分布，所以训练集目标值向量$\boldsymbol{y}$服从$\mathcal{N}(\boldsymbol{Xw},\sigma^2\boldsymbol{I})$。视训练样本$\boldsymbol{X}$非随机，当权值向量为$\boldsymbol{w}$时，训练集目标值向量$\boldsymbol{y}$的似然概率是：

$$p(\boldsymbol{y}|\boldsymbol{w})=\frac{1}{(2\pi\sigma^2)^{M/2}}\mathrm{e}^{-\frac{(\boldsymbol{y}-\boldsymbol{Xw})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{Xw})}{2\sigma^2}} \tag{5.87}$$

最大化似然概率等价于最大化它的对数：

$$\log p(\boldsymbol{y}|\boldsymbol{w})=-\frac{M}{2}\log(2\pi\sigma^2)-\frac{(\boldsymbol{y}-\boldsymbol{Xw})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{Xw})}{2\sigma^2}=-\frac{M}{2}\log(2\pi\sigma^2)-\frac{1}{2\sigma^2}\|\boldsymbol{y}-\hat{\boldsymbol{y}}\|^2 \tag{5.88}$$

对比式（5.88）和式（5.59），可发现最大化对数似然概率等价于最小化均方误差，它们都是最小化训练集目标值与预测值之间距离的平方$\|\boldsymbol{y}-\hat{\boldsymbol{y}}\|^2$。在贝叶斯框架下，真正想要最大化的并不是似然概率，而是后验概率$p(\boldsymbol{w}|\boldsymbol{y})$：

$$p(\boldsymbol{w}|\boldsymbol{y})=\frac{p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y})} \tag{5.89}$$

最大化$p(\boldsymbol{w}|\boldsymbol{y})$等价于最大化它的对数：

$$\log p(\boldsymbol{w}|\boldsymbol{y})=\log\left(\frac{p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y})}\right)=\log p(\boldsymbol{y}|\boldsymbol{w})+\log p(\boldsymbol{w})-\log p(\boldsymbol{y}) \tag{5.90}$$

假设权值向量$\boldsymbol{w}$的先验分布是$\mathcal{N}\left(\boldsymbol{0},\frac{\sigma^2}{\lambda}\boldsymbol{I}\right)$，所以$\boldsymbol{w}$的概率密度是：

$$p(\boldsymbol{w})=\frac{1}{\left(\frac{2\pi\sigma^2}{\lambda}\right)^{(n+1)/2}}\mathrm{e}^{-\frac{\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{2\sigma^2}} \tag{5.91}$$

后验概率的对数是：

$$\begin{aligned}\log p(\boldsymbol{w}|\boldsymbol{y})&=-\frac{M}{2}\log(2\pi\sigma^2)-\frac{1}{2\sigma^2}\|\boldsymbol{y}-\hat{\boldsymbol{y}}\|^2-\frac{n+1}{2}\log\frac{2\pi\sigma^2}{\lambda}-\frac{\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{2\sigma^2}-\log p(\boldsymbol{y})\\&=-\frac{M}{2\sigma^2}\left(\frac{1}{M}\|\boldsymbol{y}-\hat{\boldsymbol{y}}\|^2+\frac{\lambda}{M}\|\boldsymbol{w}\|^2\right)+C\end{aligned} \tag{5.92}$$

其中，$C=-\frac{M}{2}\log(2\pi\sigma^2)-\frac{n+1}{2}\log\frac{2\pi\sigma^2}{\lambda}-\log p(\boldsymbol{y})$，与$\boldsymbol{w}$无关。$\log p(\boldsymbol{w}|\boldsymbol{y})$括号中的部分正是式（5.78），即带$\mathcal{L}_2$正则项的均方误差损失函数。也就是说，最小化带$\mathcal{L}_2$正则项的均方误差等价于最大化后验概率$p(\boldsymbol{w}|\boldsymbol{y})$。$\boldsymbol{w}$的先验分布是$\mathcal{N}\left(\boldsymbol{0},\frac{\sigma^2}{\lambda}\boldsymbol{I}\right)$，正则化强度$\lambda$反比于$\boldsymbol{w}$各分量的方差，$\lambda$越大则权值的先验方差越小，其先验分布更集中于 0 附近。这种先验信念会导致最大后验解更靠近原点。

5.3.3　$\mathcal{L}_1$正则化

$\mathcal{L}_2$正则化之所以称为$\mathcal{L}_2$，是因为它惩罚权值向量$\boldsymbol{w}$的 2-范数的平方：$\|\boldsymbol{w}\|^2$。$\mathcal{L}_2$正则项可以写成：

$$\frac{\lambda}{M}\|\boldsymbol{w}\|^2=\frac{\lambda}{M}\sum_{i=1}^{n+1}w_i^2 \tag{5.93}$$

而$\mathcal{L}_1$正则项是权值向量$\boldsymbol{w}$的 1-范数：

$$\frac{\lambda}{M}\sum_{i=1}^{n+1}|w_i| \tag{5.94}$$

$\mathcal{L}_1$正则化和$\mathcal{L}_2$正则化都对权值分量的大小做惩罚，只不过一个惩罚的是绝对值而另一个惩罚的是平方。这个区别会导致一个有趣的效果：$\mathcal{L}_1$正则化倾向于使部分权值为 0。图 5-13 展示了带$\mathcal{L}_1$或$\mathcal{L}_2$正则项的二次函数的等高线图以及全局最小点。

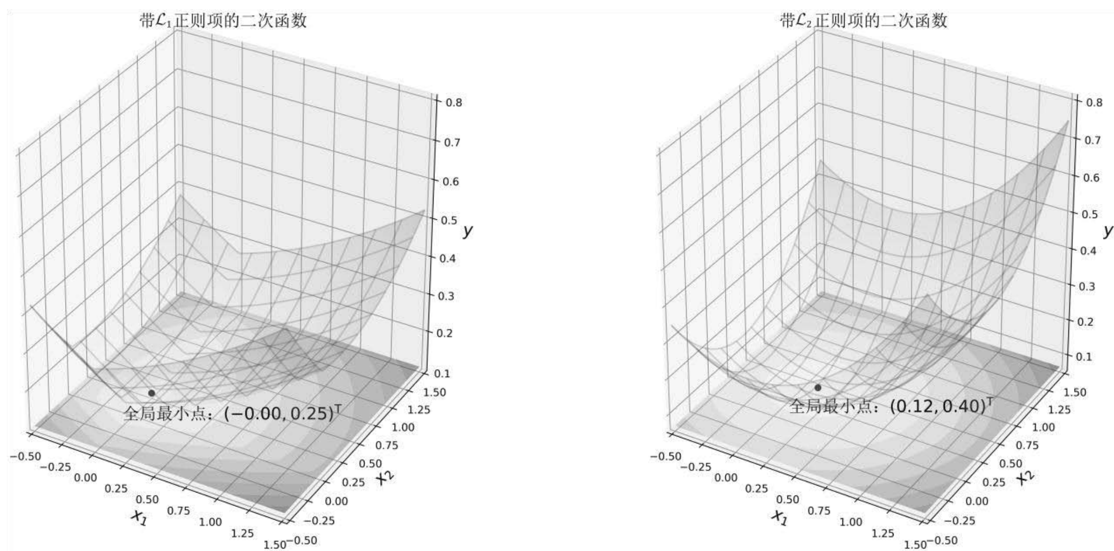


图 5-13　带$\mathcal{L}_1$或$\mathcal{L}_2$正则项的二次函数的等高线图和全局最小点

函数的等高线垂直于梯度，函数沿等高线切线的方向导数为 0。如果正则项在某一点的梯度与损失函数的梯度共线，说明损失函数的梯度垂直于正则项等高线，损失函数值沿着正则项等高线不变化。若在某点处，正则项的梯度与损失函数的梯度不共线，则损失函数的梯度与正则项的等高线不垂直，损失函数沿正则项等高线的方向导数不为 0，于是沿正则项等高线运动可以使损失函数下降。

$\mathcal{L}_2$正则项的等高线是圆，沿$\mathcal{L}_2$正则项的等高线运动时，其梯度扫过 360 度，很有可能在某点上与损失函数的梯度共线。而$\mathcal{L}_1$正则项的等高线是四边形，在同一边上的梯度相同。该梯度与损失函数的梯度共线的概率较低，于是沿着$\mathcal{L}_1$正则项的等高线运动时，大概率可以保持正则项不变而降低损失函数值，所以优化算法不大可能停留在$\mathcal{L}_1$正则项等高线的边上，而更有可能停留在其顶点，这就是$\mathcal{L}_1$正则化很有可能导致部分权值为 0 的原因。

5.4　过拟合与欠拟合

数据的真实分布是未知的，我们拥有的是从真实分布中采样的训练集，而不知道真实分布的形式。例如同一个样本集，既可以认为它采样自单一正态分布，也可以认为它采样自混合的正态分布，如图 5-14 所示。

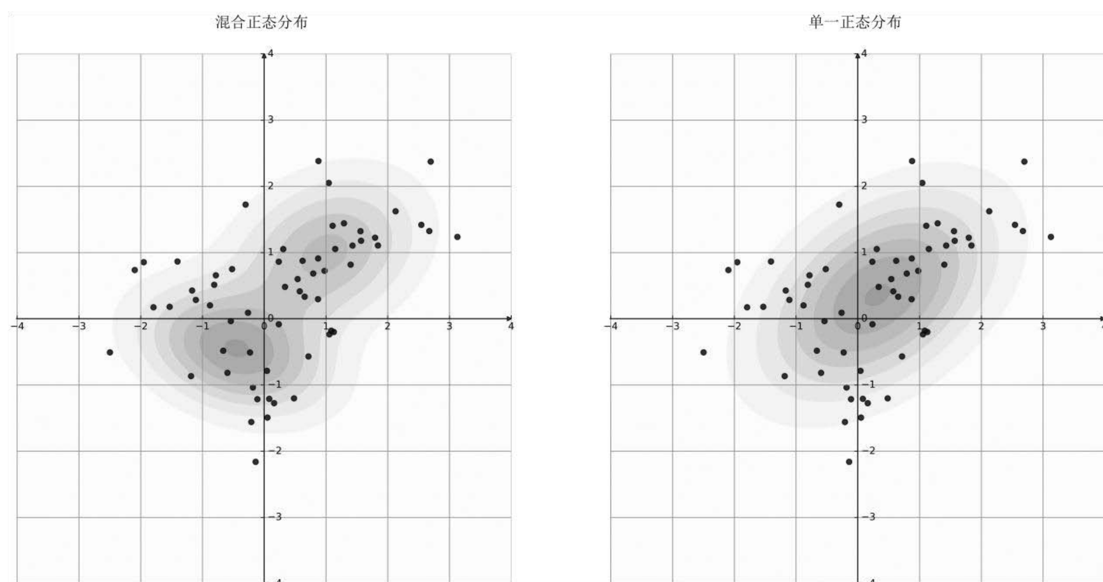


图 5-14　同一个样本集可能采样自两种不同分布

当真实分布的形式未知时，我们需要做先验的假设。例如，线性模型只能产生超平面分界面，采用线性模型就是假设真实的两类数据是线性可分的。如果假设真实数据不是线性可分的，则可以采用非线性模型，如图 5-15 所示。

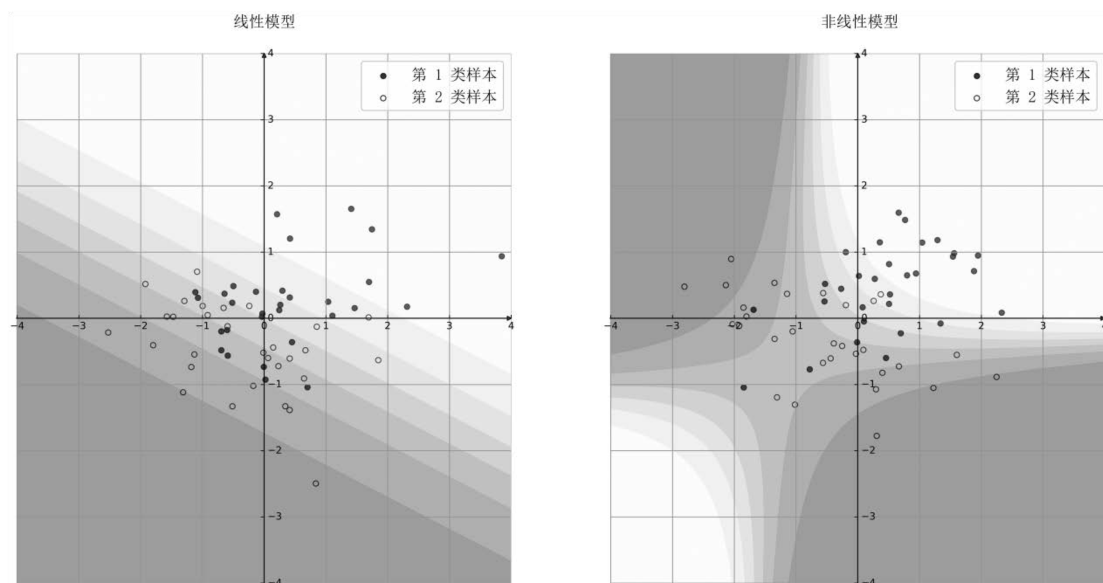


图 5-15　不同的先验假设采用不同的模型

有时即便知道真实数据线性不可分，但仍采用线性模型，因为真实分布可能非常复杂，并且我们对其一无所知，这时无法做出恰当的先验假设。同时训练样本可能很稀少，一旦模型过于复杂，则很可能学习不到数据的真实分布。这种情况下采用简单的线性模型反倒能得到更好的效果，如图 5-16 所示。

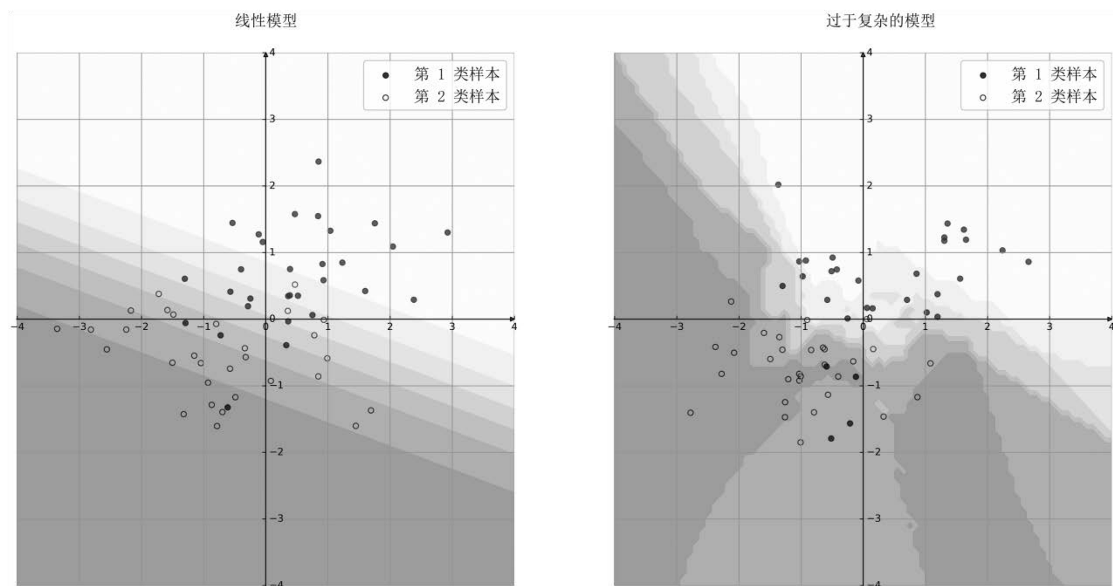


图 5-16　分布复杂而训练样本稀少

图 5-16 右侧对数据的真实分布做了过于复杂的假设，采用自由度过高的模型，导致预测新样本时表现差，这种情况称为过拟合（over-fitting）。但是如果先验假设过于简单，采用自由度过低的模型，也会因为学习不到数据的真实分布而导致预测效果差，这种情况称为欠拟合（under-fitting）。

模型种类、参数规模、正则化方法及强度等因素都影响模型的自由度。高自由度导致低偏置高方差，但由于数据的真实分布未知，无偏于一个假设的期望并不一定就好。低自由度降低方差而增大偏置，偏离假设的期望不一定就坏。

第 2 章提到应该在独立的测试集上评价模型，是因为如果我们做了不恰当的先验假设，并选用了高自由度的模型，那么模型偏置小会导致过拟合，这时训练集上的指标会过于乐观，可是在独立的测试集上的指标就会很差，从而暴露过拟合现象。训练集和测试集上的评价指标随模型自由度变化的典型情形如图 5-17 所示。

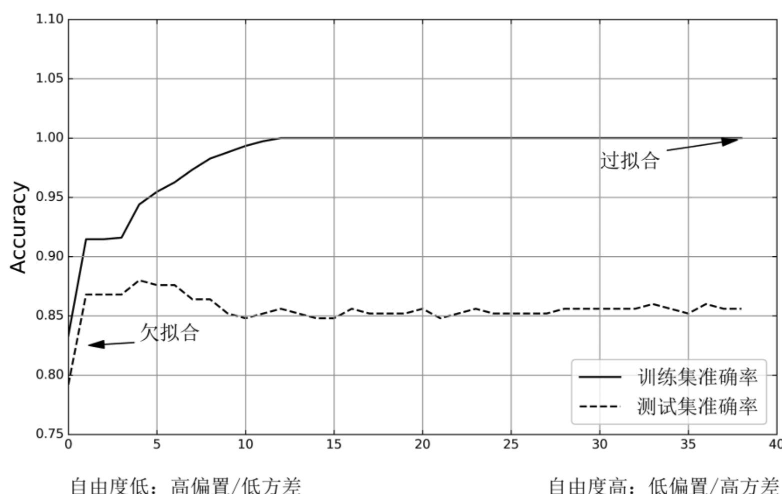


图 5-17　训练集和测试集的正确率随模型自由度的变化

模型调优的通常做法是：首先，将带标签的样本分成训练集、验证集和测试集。通过调整各种超参数，控制自由度从低到高变化，在训练集上多次训练模型。同时监控训练集和验证集上的评价指标，在欠拟合和过拟合之间寻找最佳自由度。注意，验证集不参与训练，只用来监控偏置–方差权衡。找到合适的自由度，并用这套超参数在训练集上训练模型后，在测试集上评价该模型。测试集不参与模型训练和选择，在它之上获得的评价指标是客观的。

5.5　运用$\mathcal{L}_2$正则化训练逻辑回归

训练集是$S=\{\boldsymbol{x}^i,y^i\}_{i=1}^M$，给训练样本$\boldsymbol{x}^i$（$i=1,\cdots,M$）增加一维常量 1，从而将偏置$b$纳入权值向量$\boldsymbol{w}$。$y^i$（$i=1,\cdots,M$）是 1/0 编码的类别标签。带$\mathcal{L}_2$正则项的交叉熵损失函数是：

$$\operatorname{loss}(\boldsymbol{w})=-\frac{1}{M}\sum_{i=1}^M\left(y^i\ \log\frac{1}{1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}+(1-y^i)\log\frac{1}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^i}}\right)+\frac{\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{M} \tag{5.95}$$

我们将训练样本视为非随机，只将训练标签视为随机，根据式（2.25），$\operatorname{loss}(\boldsymbol{w})$的第一项是$-\frac{1}{M}\log p(S|\boldsymbol{w})$，它与训练集$S$的对数似然概率的相反数成正比。观察到训练集$S$的前提下，权值是$\boldsymbol{w}$的对数后验概率是：

$$\log p(\boldsymbol{w}|S)=\log p(S|\boldsymbol{w})+\log p(\boldsymbol{w})-\log p(S) \tag{5.96}$$

假设$\boldsymbol{w}$的先验分布是$\mathcal{N}(\boldsymbol{0},(1/2\lambda)\boldsymbol{I})$，则$\boldsymbol{w}$的概率密度是：

$$p(\boldsymbol{w})=\frac{1}{(\pi/\lambda)^{(n+1)/2}}\mathrm{e}^{-\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \tag{5.97}$$

于是对数后验概率是：

$$\log p(\boldsymbol{w}|S) = \sum_{i=1}^{M}\left(y^i \log\frac{1}{1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}x^i}} + (1-y^i)\log\frac{1}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}x^i}}\right) - \lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + C \qquad (5.98)$$

C是与$\boldsymbol{w}$无关的常数。结合式（5.95），最大化$\log p(\boldsymbol{w}|S)$就是最小化$\mathrm{loss}(\boldsymbol{w})$。梯度下降法需要计算$\mathrm{loss}(\boldsymbol{w})$对$\boldsymbol{w}$的梯度，也就需要计算$\mathrm{loss}(\boldsymbol{w})$对$\boldsymbol{w}$每个分量的偏导数。$\mathrm{loss}(\boldsymbol{w})$是交叉熵和$\mathcal{L}_2$正则项之和，它对$w_j$的偏导数也是这两项对$w_j$的偏导数之和：

$$\frac{\partial\left(\frac{\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{M}\right)}{\partial w_j} = \frac{\partial\left(\frac{\lambda}{M}\sum_{j=1}^{n+1} w_j^2\right)}{\partial w_j} = \frac{2\lambda w_j}{M} \qquad (5.99)$$

根据式（3.36）有：

$$\frac{\partial}{\partial w_j}\left(-\frac{1}{M}\log p(S|\boldsymbol{w})\right) = -\frac{1}{M}\sum_{i=1}^{M}\left(y^i - p^i(\boldsymbol{w})\right)x_j^i \qquad (5.100)$$

其中，$p^i(\boldsymbol{w})$是逻辑回归对第i个训练样本的预测概率：

$$p^i(\boldsymbol{w}) = \frac{1}{1+\mathrm{e}^{-\boldsymbol{w}^{\mathrm{T}}x^i}} \qquad (5.101)$$

所以$\mathrm{loss}(\boldsymbol{w})$的梯度是：

$$\nabla\mathrm{loss}(\boldsymbol{w}) = -\frac{1}{M}\sum_{i=1}^{M}\left(y^i - p^i(\boldsymbol{w})\right)\begin{pmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_n^i \end{pmatrix} + \frac{2\lambda}{M}\boldsymbol{w} \qquad (5.102)$$

于是，带$\mathcal{L}_2$正则化的逻辑回归训练的伪代码如下：

```
w^0 ← 随机初始化
t ← 0
while ‖∇loss(w^t)‖ ⩾ ε:
    w^{t+1} ← w^t + η · ( (1/M) Σ_{i=1}^{M} (y^i − p^i(w^t)) (x_0^i, x_1^i, ⋮, x_n^i)^T − (2λ/M) w^t )
    t ← t + 1
return w^t
```

可以将算法的更新式写成：

$$\boldsymbol{w}^{t+1} \leftarrow \left(1-\frac{2\eta\lambda}{M}\right)\boldsymbol{w}^t + \frac{\eta}{M}\sum_{i=1}^{M}\left(y^i - p^i(\boldsymbol{w}^t)\right)\begin{pmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_n^i \end{pmatrix} \qquad (5.103)$$

从式（5.103）看出：更新式先对$\boldsymbol{w}^t$进行压缩，压缩因子是$\left(1-\frac{2\eta\lambda}{M}\right)$，这就是$\mathcal{L}_2$正则化将权值拉向原点的原因。

5.6　运用$\mathcal{L}_2$正则化训练逻辑回归的 Python 实现

本节，我们为第 3 章实现的逻辑回归模型添加正则化。我们实现$\mathcal{L}_1$和$\mathcal{L}_2$两种正则化，正则化方法和正则化强度作为构造函数的参数，代码如下：

```
from optimizer import *
import numpy as np


class LogisticRegression_reg:

    def __init__(self, optimizer, iterations = 50000, regularization = "l2", reg_lambda = 0.01):

        assert(optimizer is not None)

        self.optimizer = optimizer
        self.iterations = iterations # 迭代次数
        self.regularization = regularization # 正则化方法
        self.reg_lambda = reg_lambda # 正则化强度系数

    def train(self, x, y):
        """
        x 为矩阵，形状是 n_samples * n_features，每一行为一个样本
        y 为矩阵，形状是 n_samples * 1，元素为样本的标签，正类为 1，负类为 0
        """

        # 在 x 最前面添加一列常数 1，作为偏置值的输入，以简化公式
        x = np.mat(np.c_[[1.0] * x.shape[0], x])

        # 根据 x 的列数（特征数）随机初始化权值，此时偏置值纳入了权值向量，相当于第一个权值
        # 权值向量为 n_features + 1 维向量，每个分量以 0 均值、0.01 标准差的正态分布初始化
        self.weights = np.mat(np.random.normal(0, 0.01, size=x.shape[1])).T

        for i in range(self.iterations):

            # 计算当前模型对训练集样本的输出
            p = self.predict(x, False)

            gradient = -x.T * (y - p) / x.shape[0] # 交叉熵损失对模型参数的梯度
```

```
        # 正则项的的梯度
        if self.regularization == "l2":
            gradient += 2 * self.reg_lambda * self.weights / x.shape[0]
        elif self.regularization == "l1":
            gradient += self.reg_lambda * np.sign(self.weights) / x.shape[0]

        self.weights += self.optimizer.delta(gradient) # 更新模型参数

        # 评估当前模型并打印训练信息
        if i % 100 == 0:
            # 交叉熵损失
            cross_entropy = (-y.T * np.log(p) - (1.0 - y).T * np.log(1 - p)) / y.shape[0]

            # 正确率
            accuracy = np.sum(((p > 0.5).astype(np.int) == y).
                astype(np.int)) / y.shape[0]

            print("迭代: {:d}, 交叉熵: {:.6f}, 正确率: {:.2f}%".format(
                i + 1, cross_entropy[0, 0], accuracy * 100))


def predict(self, x, augment = True):
    """
    预测函数。x 为矩阵, 形状是 n_samples * n_features, 每一行为一个样本
    augment 参数指示是否要在特征矩阵前添加一列常量 1
    """
    # 在 x 最前面添加一列常数 1, 作为偏置值的输入
    if augment:
        x = np.mat(np.c_[[1.0] * x.shape[0], x])

    a = -np.matmul(x, self.weights)
    a[a > 1e2] = 1e2  # 防止数值过大
    p = 1.0 / (1.0 + np.power(np.e, a))

    # 剪裁概率值, 保证其为合法的概率值
    p[p >= 1.0] = 1.0 - 1e-10
    p[p <= 0.0] = 1e-10

    return p
```

新添的代码段根据不同的正则化方法，将正则项的梯度加到交叉熵的梯度之上。对于$\mathcal{L}_1$正则项，求偏导数时，若参数为正，则取 1；若参数为负，则取−1，再乘上正则化强度并除以样本个数。

```
# 正则项的梯度
if self.regularization == "l2":
    gradient += 2 * self.reg_lambda * self.weights / x.shape[0]
elif self.regularization == "l1":
    gradient += self.reg_lambda * np.sign(self.weights) / x.shape[0]
```

接下来，我们使用带正则项的逻辑回归建模鸟类生态类群问题。读者可自行调节正则化方法和强度，观察它们对模型参数以及模型效果的影响。

```
import pandas as pd
import numpy as np
from optimizer import *
from lr_reg import LogisticRegression_reg
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score


bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)

# 根据标签是否属于"SW", "W", "R"三类，构造二分类 1/0 标签
bird["type"] = bird.type.apply(lambda t: t in ["SW", "W", "R"]).astype(np.int)

data = bird.values
# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集
train_x = np.mat(data[:300,:-1])
train_y = np.mat(data[:300,-1]).T

# 其余样本作为测试集
test_x = np.mat(data[300:,:-1])
test_y = np.mat(data[300:,-1]).T

# 构造逻辑回归对象，优化器为原始梯度下降，学习率 0.01，使用 L2 正则化，正则化强度 0.01
lr = LogisticRegression_reg(Gradient(0.01), regularization = "l2", reg_lambda = 0.01)

# 在训练集上训练
lr.train(train_x, train_y)

# 对测试集进行预测
p = lr.predict(test_x) # 模型预测的正类概率
pred = (p > 0.5).astype(np.int) # 以 0.5 为阈值时，模型预测的类别

print("正确率：{:.2f}%，查准率：{:.2f}%，查全率：{:.2f}%，ROC 曲线下面积：{:.3f}".format(
```

```
        accuracy_score(test_y, pred) * 100,
        precision_score(test_y, pred) * 100,
        recall_score(test_y, pred) * 100,
        roc_auc_score(test_y.A, p)))
```

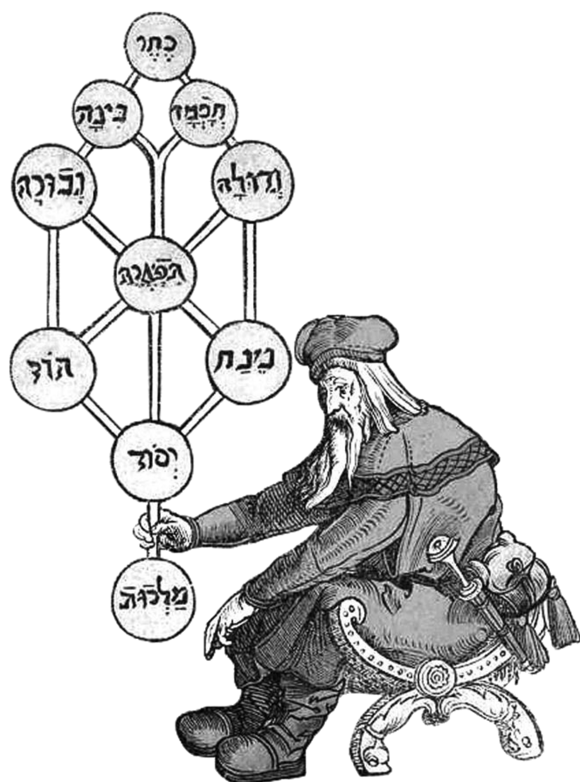
5.7 小结

通过本章，读者应该对模型自由度、偏置–方差权衡、正则化、过拟合与欠拟合等概念有了较深刻的理解。除本书涉及的线性回归、逻辑回归、神经网络和深度学习外，这些概念可应用到机器学习领域的其他众多模型。在那些模型中，这些概念可能没有精确的数学定义，但是原理仍然是共通的。例如决策树模型，树的最大深度是一个超参数。若最大深度较大，则决策树的自由度高、偏置较低而方差较高，易发生过拟合；若最大深度较小，则决策树的自由度低、偏置较高而方差较低，易发生欠拟合。

除正则化强度外，各类模型和训练算法还有数量众多的超参数，另外，模型的自由度还受很多隐含因素的影响。熟悉各种超参数，理解它们对自由度的作用，并能够调节它们控制自由度，是机器学习工程师的基本素养。

第二部分

神经网络



第 6 章

神经网络

6

神经元对输入向量施加仿射函数和激活函数，它的输出只能沿着权值向量的方向变化，所以神经元只能形成垂直于权值向量的超平面分界面。逻辑回归模型就是一个神经元。神经元（线性模型）的结构、特性和训练是本书第一部分的内容。

从本章开始，我们进入神经网络领域。神经网络（neural network）把多个神经元连接成网络。所谓“连接”，就是把一个神经元的输出当作另一些神经元的输入。连接方式众多，本章只介绍最简单的一种：多层全连接神经网络。

将多个神经元连接成网络能产生非线性分界能力，这种能力来自神经元之间的合作和非线性的激活函数。除了（逻辑回归的）Logistic 函数，还有其他很多种激活函数。本章介绍最常用、最具代表性的几种激活函数以及它们的性质。最后，本章由多分类问题引入 SoftMax 函数，它将神经网络的多个输出转化为多分类概率分布。

6.1 合作的神经元

令逻辑回归的权值向量和偏置是$\boldsymbol{w}$和b，它先对输入向量$\boldsymbol{x}$施加仿射函数$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$，再施加 Logistic 函数，以 Logistic 函数的输出为正类的概率。因为 Logistic 函数单调递增，所以逻辑回归其实是根据仿射函数值$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$的大小来预测类别的。仿射函数的等值面是垂直于权值向量$\boldsymbol{w}$的超平面，所以逻辑回归具有超平面分界面。如果以 0.5 为概率阈值，那么分界面就是$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$。这个分界面的法向量是$\boldsymbol{w}$，不同$\boldsymbol{w}$的逻辑回归模型如图 6-1 所示。

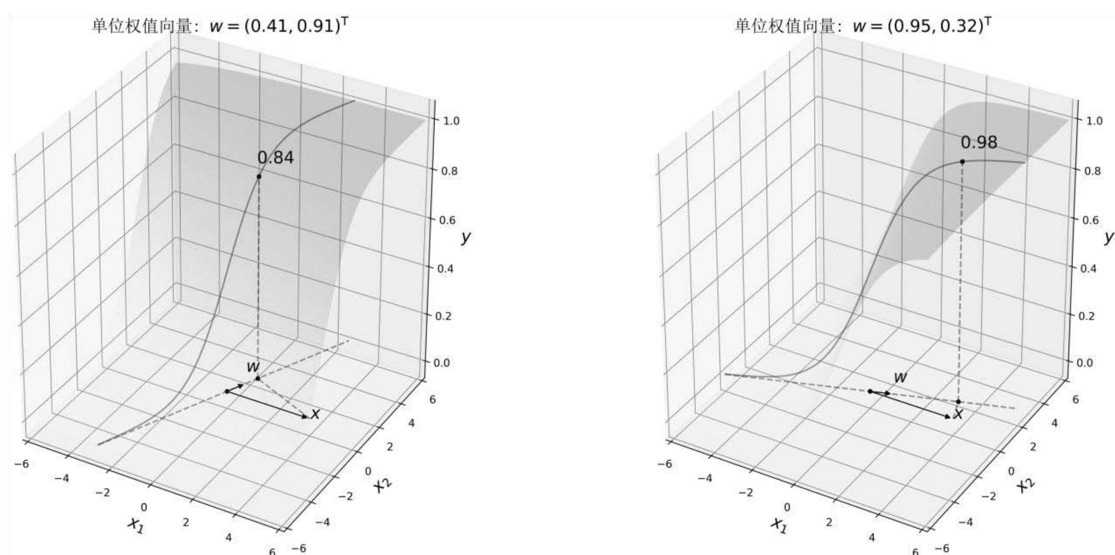


图 6-1　不同权值向量的逻辑回归模型

现在我们把多个逻辑回归模型相加，构造一个复合模型：

$$f(\boldsymbol{x}) = \sum_{i=1}^{k} \frac{1}{1+\mathrm{e}^{-b^i-(w^i)^{\mathrm{T}}x}} \tag{6.1}$$

可以用神经元示意图表示该复合模型。当$\boldsymbol{x}$是 2 维，$k = 3$时，如图 6-2 所示。

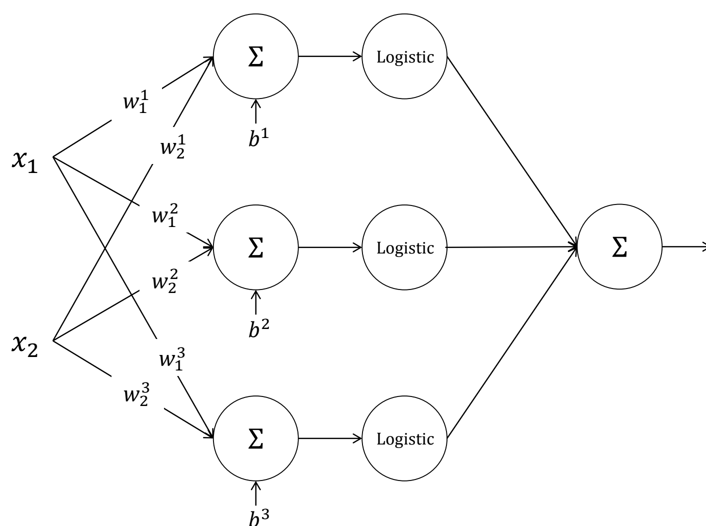


图 6-2　多个逻辑回归之和

以k个逻辑回归模型的权值向量作为行，构造$k \times n$的权值矩阵$\boldsymbol{W}$，以k个偏置为分量构造偏置向量$\boldsymbol{b}$，于是可以将式（6.1）写成矩阵形式：

$$f(\boldsymbol{x}) = (1 \quad \cdots \quad 1)_{1\times k} \cdot \mathrm{Logistic}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) \tag{6.2}$$

$\mathrm{Logistic}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$对向量$\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}$的每个分量施加 Logistic 函数。用$1\times k$的全 1 矩阵乘以$\mathrm{Logistic}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$，等于将其全部分量相加。当$\boldsymbol{x}$是 2 维，$k=3$时，式（6.2）的典型函数图像如图 6-3 所示。

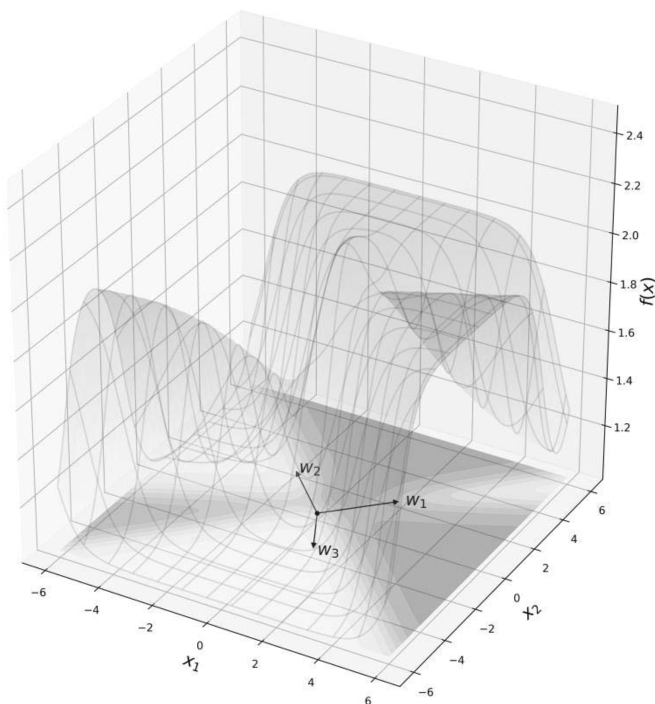


图 6-3　三个逻辑回归模型之和

在图 6-3 中，三个逻辑回归模型的权值向量分别指向不同的方向。每个逻辑回归模型沿它的权值向量的方向抬高函数值，沿相反的方向压低函数值。将三个模型的值相加就得到图中复杂的图像。图中的箭头就是三个逻辑回归模型的权值向量。复杂的等高线说明复合模型具有非线性分界能力。Logistic 函数是形成非线性分界能力所必不可少的，如果从式（6.2）中去掉 Logistic 函数，则变成：

$$f(\boldsymbol{x}) = (1 \quad \cdots \quad 1)_{1\times k} \cdot (\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b \tag{6.3}$$

其中：

$$\boldsymbol{w}=\begin{pmatrix}\sum_{i=1}^{k} w_1^i \\ \vdots \\ \sum_{i=1}^{k} w_n^i\end{pmatrix} \tag{6.4}$$

w_j^i是第i个逻辑回归模型的权值向量的第j个分量，$b=\sum_{i=1}^{k} b^i$是k个偏置之和。式（6.3）表明：去掉 Logistic 函数后的复合模型退化成了仿射函数，如图 6-4 所示。

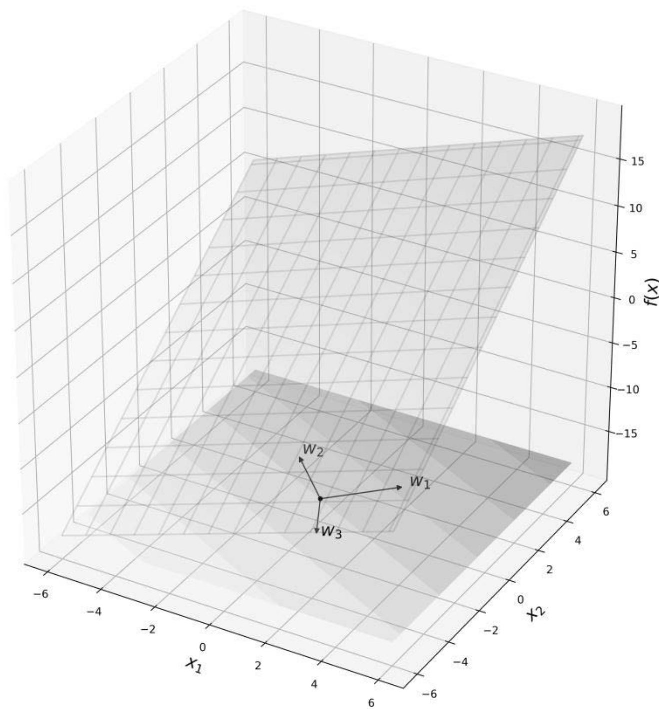


图 6-4　去掉 Logistic 函数的复合模型

Logistic 函数将输出压缩在(0, 1)区间。逻辑回归模型的输出沿着权值向量的方向扬起，趋近于 1，沿相反的方向压低，趋近于 0。我们可以将扬起的一端称为激活端，将压低的一端称为抑制端。每一个逻辑回归模型的抑制端趋近于 0，参与加和时不影响另一个模型沿着该方向扬起，这就允许多个逻辑回归模型各管一方，各司其职，形成合作。

如果去掉 Logistic 函数，模型的激活端可以高到正无穷，而抑制端可以低到负无穷。那么一个模型的抑制端就有可能抵消另一个模型的激活端，使它们抬高输出的努力成为徒劳，无法合作形成复杂分界面，如图 6-5 所示。

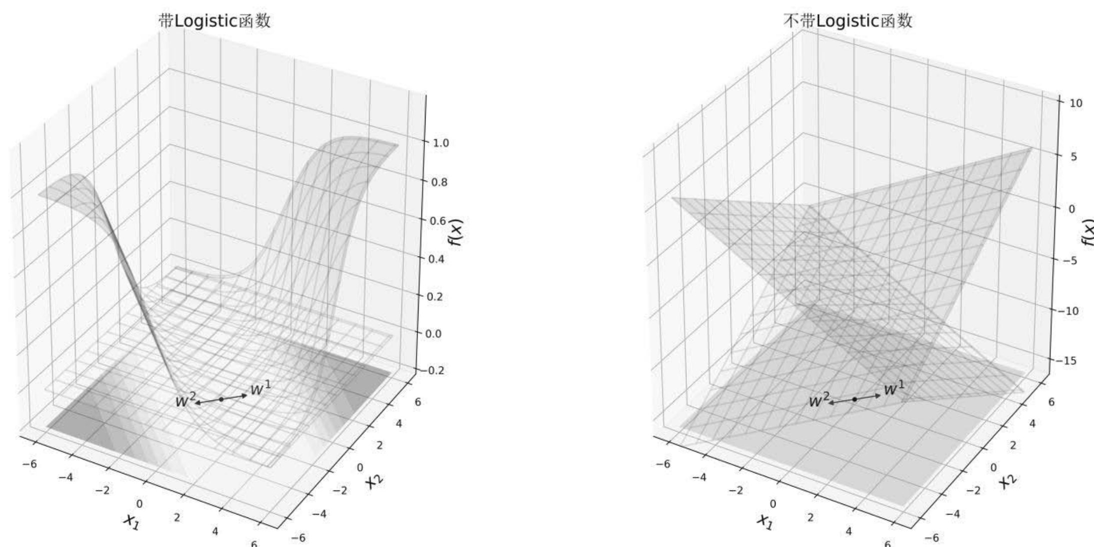


图 6-5　输出不加限制，则模型互相抵消，无法合作

在图 6-5 左图中，两个逻辑回归模型朝相反的方向激活，它们的抑制端趋近于 0，相加后不抵消对方的激活端。右图中的两个模型没有 Logistic 函数，于是每个模型的抑制端抵消了对方的激活端。正如式（6.3）表明的那样，这两个模型之和是一个仿射函数。

6.2　多层全连接神经网络

对“逻辑回归之和”复合模型进行改进：将简单的加和变成加权求和再加偏置，也就是对多个逻辑回归的输出施加仿射函数，为了使结果能被解释为概率，再对这个仿射函数的输出施加 Logistic 函数，将输出控制在(0, 1)区间。这其实就是将多个逻辑回归的输出当作另一个逻辑回归的输入。当$\boldsymbol{x}$是 2 维，$k = 3$时，这个复合模型如图 6-6 所示。

这个复合模型就是一个简单的多层全连接神经网络，每一个逻辑回归就是一个神经元。Logistic 函数被称为激活函数（activation function）。权值和偏置的上标表示层数，第一脚标表示层内神经元的编号。权值的第二脚标表示该连接所连的上一层神经元的编号，例如$w_{1,2}^2$表示这个权值是第二层的第一个神经元与上一层第二个输出之间的连接的权重。

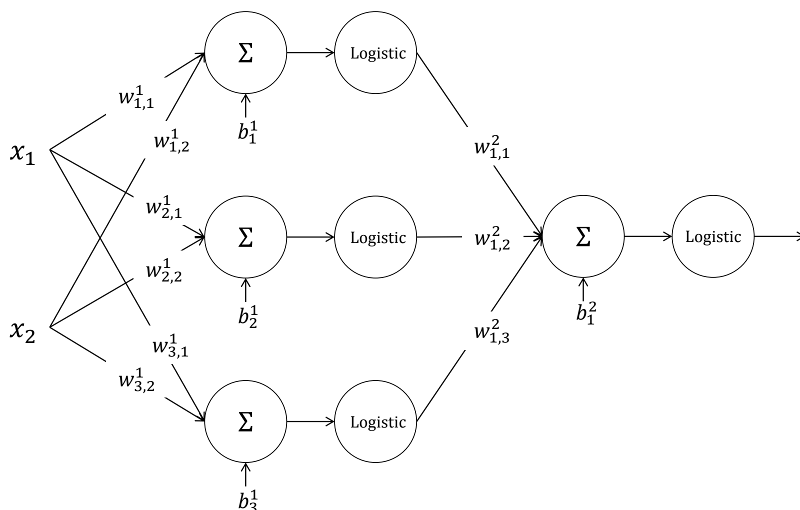


图 6-6　将多个逻辑回归的输出当作另一个逻辑回归的输入

多层全连接神经网络又称为多层感知机，它的神经元被组织成层状，层数不限。输入向量的每一个分量连接到第一层的每一个神经元，前一层每个神经元的输出连接到下一层每个神经元的输入。最后一层神经元的输出是网络的输出，称为输出层，其他层称为隐藏层，因为它们的输出在网络外部不可见。图 6-6 中的复合模型是一个 2 个输入、1 个隐藏层、1 个输出层的多层全连接神经网络，其唯一的隐藏层包含三个神经元。

本书用上标表示层的编号，最接近输入的层是第 1 层，第一脚标是层内神经元的编号，第二脚标是神经元输入的编号。例如第k层的第i个神经元的权值向量是$\boldsymbol{w}_i^k$。$\boldsymbol{w}_i^k$的第j个分量，即该神经元对第j个输入的权值是$w_{i,j}^k$。这个神经元的偏置是b_i^k，它的仿射函数的输出用a_i^k表示，激活函数的输出用x_i^k表示。之所以用字母x表示输出，是因为这层的输出也是下一层的输入。输入向量$\boldsymbol{x}$的维数是n^0，每一个分量为x_i^0。第k层神经元的数量用n^k表示。

因为网络中层与层之间是全连接的，所以第k层的每个神经元的输入数量等于前一层的神经元的个数n^{k-1}。以第k层神经元的权值向量作为行，可构造$n^k \times n^{k-1}$的权值矩阵$\boldsymbol{W}^k$。以第k层神经元的偏置为分量，可构造偏置向量$\boldsymbol{b}^k$。于是图 6-6 中的复合模型就可以表示成：

$$f(\boldsymbol{x}) = \text{Logistic}(\boldsymbol{W}^2 \cdot \text{Logistic}(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) \tag{6.5}$$

式（6.5）中的 Logistic 函数是对向量的每一个分量施加的。$\boldsymbol{x}$是 2 维向量，$\boldsymbol{W}^1$是 3×2 矩阵。$\boldsymbol{b}^1$是 3 维向量，$\boldsymbol{W}^2$是 1×2 矩阵，$\boldsymbol{b}^2$是 1 维向量，即标量。整个$f(\boldsymbol{x})$是一个$\mathbb{R}^3 \to \mathbb{R}^1$的函数。$f(\boldsymbol{x})$的典型图像如图 6-7 所示。

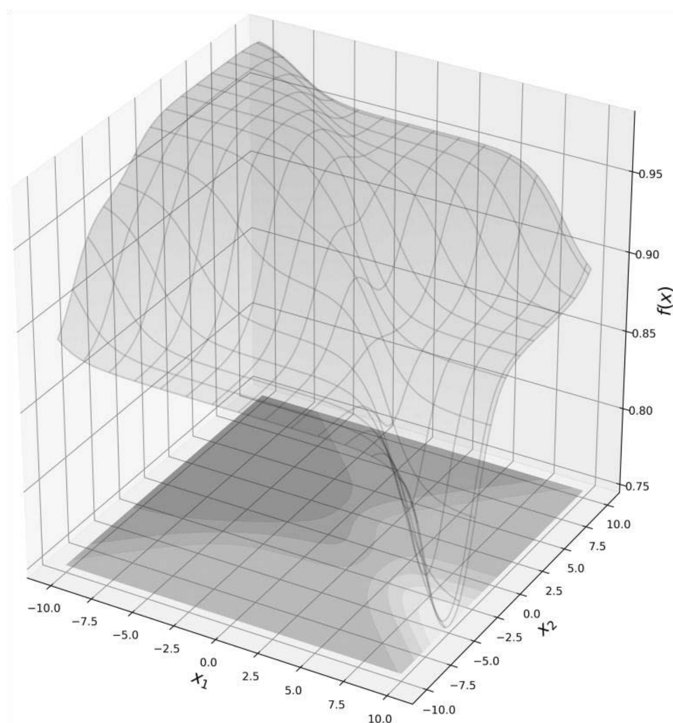


图 6-7　2 个输入、1 个输出的多层全连接神经网络的典型图像

多层全连接神经网络可以有多个隐藏层，输出层也可以有多个神经元。图 6-8 展示了一个例子，该网络接受 2 个输入，第一隐藏层和第二隐藏层各有 3 个神经元，输出层有 2 个神经元，即网络的输出是一个 2 维向量。

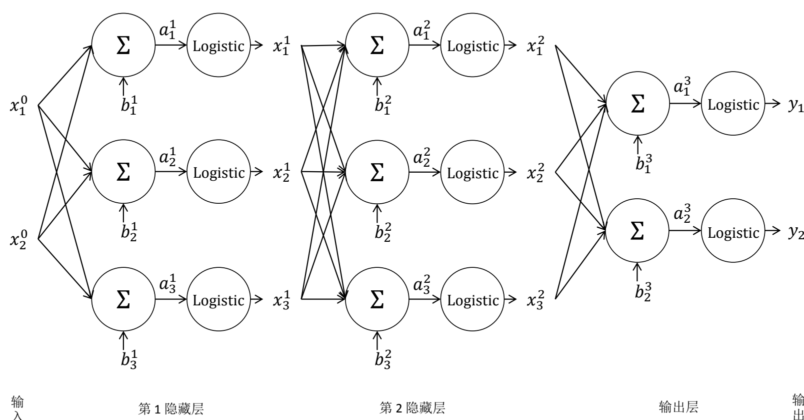


图 6-8　多层全连接神经网络

网络输入和输出的维数应根据问题而定，隐藏层数以及每个隐藏层的神经元数可任意设置。激活函数也不限于 Logistic 函数，每一层甚至每个神经元都可以采用不同的激活函数。例如，如果目的是拟合某个映射，那么将输出限制在(0, 1)区间显然是不合适的，这种情况下输出层神经元可以采用恒等函数作为激活函数，相当于没有激活函数。多层全连接神经网络的结构非常灵活，但它执行的无非是一个映射，图 6-8 中的神经网络执行的映射是：

$$f(\boldsymbol{x}) = \text{Logistic}(\boldsymbol{W}^3 \cdot \text{Logistic}(\boldsymbol{W}^2 \cdot \text{Logistic}(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) \tag{6.6}$$

式（6.6）中各个权值矩阵和偏置向量具有恰当的维数。注意激活函数的作用，若去掉 Logistic 函数，则式（6.6）退化为：

$$f(\boldsymbol{x}) = \boldsymbol{W}^3\boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{W}^3\boldsymbol{W}^2\boldsymbol{b}^1 + \boldsymbol{W}^3\boldsymbol{b}^2 + \boldsymbol{b}^3 = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{6.7}$$

式（6.7）是仿射映射，下一章会详细介绍。多层全连接神经网络是一种简单的神经网络，它不包含反馈，连接只存在于层与层之间，层内部的神经元之间没有连接。多层全连接神经网络的结构取决于：输入和输出的维数、隐藏层的数量、各隐藏层的神经元数量以及每个神经元的激活函数。结构确定后，网络的行为取决于各层的权值矩阵和偏置向量，它们是待训练的模型参数。

6.3 激活函数

在神经网络中，神经元对仿射函数的输出施加的单调函数称为激活函数。逻辑回归模型作为神经元，它的激活函数就是我们熟悉的 Logistic 函数。前文介绍过，激活函数对于形成非线性分界面功不可没。若去掉激活函数，则无论有多少层、多少神经元，网络都将退化成为仿射映射。除了 Logistic 函数，还有很多种激活函数，本节介绍最常用、最有代表性的几种并讨论它们的性质。在不引起混淆的情况下，每节都用$f(x)$表示该节讲解的激活函数。

6.3.1 Linear

最简单的激活函数是线性函数（linear function）：

$$f(x) = \text{Linear}(x) = x \tag{6.8}$$

线性函数也称恒等函数（identity function）。线性函数的导数是：

$$f'(x) \equiv 1 \tag{6.9}$$

线性函数及其导数的图像如图 6-9 所示。

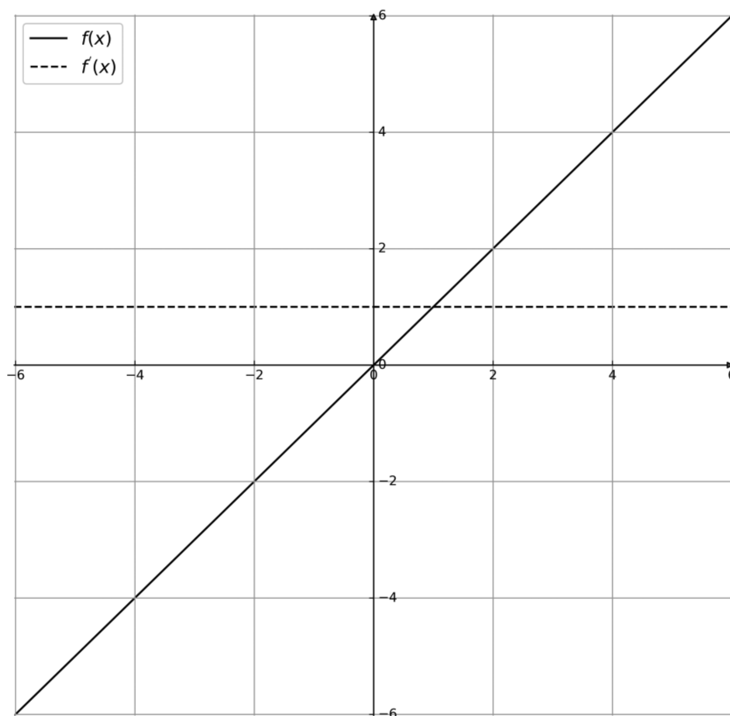


图 6-9　线性函数及其导数的图像

神经元以线性函数为激活函数，就相当于没有激活函数。

6.3.2　Logistic

Logistic 函数我们已经非常熟悉了，它是逻辑回归的固有组成。当逻辑回归模型被视为神经元时，Logistic 函数就是它的激活函数。再次将 Logistic 函数写出：

$$f(x)=\text{Logistic}(x)=\frac{1}{1+\mathrm{e}^{-x}} \tag{6.10}$$

Logistic 函数的导数是：

$$f'(x)=\frac{\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^2}=\frac{1}{1+\mathrm{e}^{-x}}\cdot\frac{\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}}=f(x)\big(1-f(x)\big) \tag{6.11}$$

式（6.11）表明，在已经计算出 Logistic 函数在x的值之后，也很容易计算它在x的导数。在训练时，这个性质可以节省计算量。Logistic 函数及其导数的图像如图 6-10 所示。

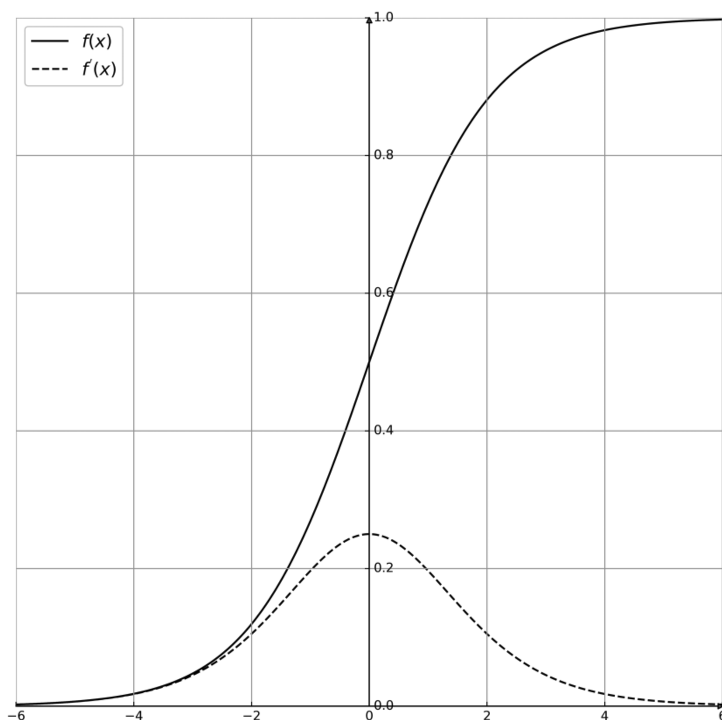


图 6-10 Logistic 函数及其导数的图像

Logistic 函数在正无穷趋近于 1，在负无穷趋近于 0，其图像整体呈 S 形，故这条曲线又称为 sigmoid 曲线。Logistic 函数的导数在原点最大，为 0.25。在原点周围的小邻域内，Logistic 函数接近线性函数。Logistic 函数的导数向正负两侧衰减，所以 Logistic 函数在两侧远端的图像接近水平线，也称为在正负两侧远端“饱和”。Logistic 函数在“饱和”区域的导数接近 0，这对神经元的训练不利。可以通过加入一个参数α来控制 Logistic 函数在原点的斜率：

$$f(x;\alpha) = \text{Logistic}(x;\alpha) = \frac{1}{1+\mathrm{e}^{-\alpha x}} \tag{6.12}$$

$f(x;\alpha)$的导数是：

$$f'(x;\alpha) = \frac{\alpha \mathrm{e}^{-\alpha x}}{(1+\mathrm{e}^{-\alpha x})^2} = \alpha f(x;\alpha)\big(1-f(x;\alpha)\big) \tag{6.13}$$

$f(x;\alpha)$在原点的导数是$\frac{\alpha}{4}$，通过参数α可调整$f(x;\alpha)$在原点的斜率，如图 6-11 所示。

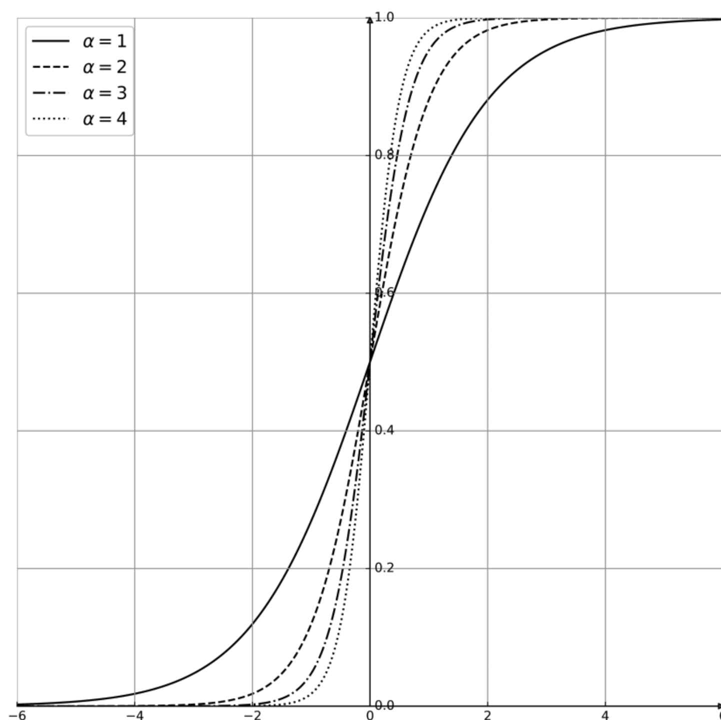


图 6-11　不同α的 Logistic 函数的图像

当α趋近于正无穷时，Logistic 函数趋近于阶跃函数（step function）：

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geqslant 0 \end{cases} \tag{6.14}$$

阶跃函数在原点不可导，甚至不连续。现代神经网络一般不采用阶跃函数作为激活函数。

6.3.3　Tanh

Tanh 又称双曲正切函数，它与 Logistic 函数相似，但值域是$(-1,1)$：

$$f(x) = \mathrm{Tanh}(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}} \tag{6.15}$$

对 Tanh 变形，可发现它与 Logistic 函数的联系：

$$f(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}} = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}} = \frac{1}{1+\mathrm{e}^{-2x}} - \frac{\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}} = 2 \cdot \mathrm{Logistic}(x;2) - 1 \tag{6.16}$$

Tanh将Logistic(x; 2)缩放为2倍，再向下移动1。Logistic(x; 2)的值域是(0, 1)，缩放后是(0, 2)，下移后 Tanh 的值域是(−1, 1)。Logistic(x; 2)在原点的值是 0.5，缩放后是 1，下移后 Tanh 在原点的值是 0。Tanh 的导数也容易求得：

$$f'(x) = 4 \times \text{Logistic}(x;2)\big(1 - \text{Logistic}(x;2)\big) = 4 \times \frac{1+f(x)}{2} \cdot \frac{1-f(x)}{2} = 1 - f(x)^2 \quad (6.17)$$

Tanh 函数及其导数的图像如图 6-12 所示。

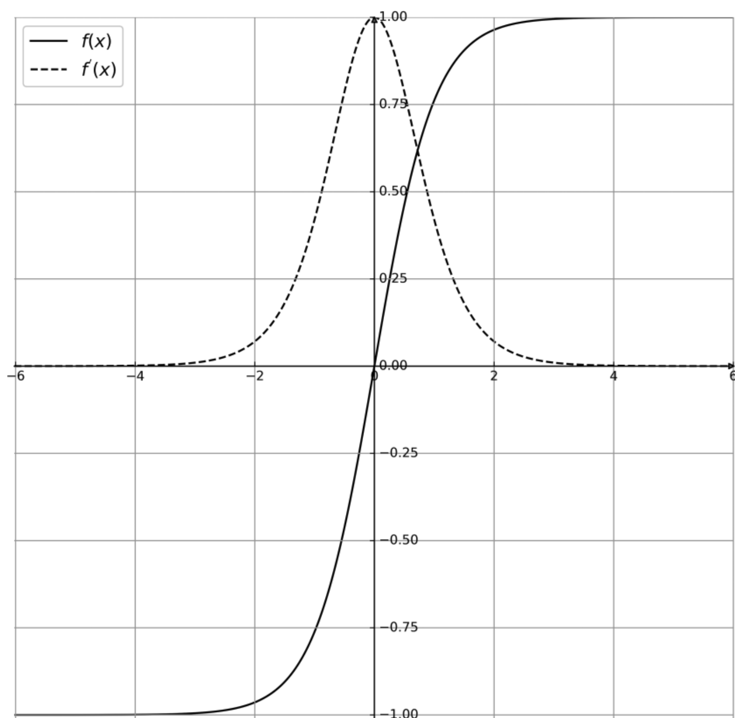


图 6-12 Tanh 函数及其导数的图像

Tanh 的图像也是 sigmoid 曲线，在正无穷趋近于 1，在负无穷趋近于 0，在原点的导数为 1。Tanh 在原点周围接近直线，在正负远端饱和，接近水平线。同样可以通过给x添加系数α来调整 Tanh 在原点的斜率：

$$f(x;\alpha) = \text{Tanh}(x;\alpha) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} = 2 \cdot \text{Logistic}(x;2\alpha) - 1 \quad (6.18)$$

不同α的 Tanh 函数的图像如图 6-13 所示。

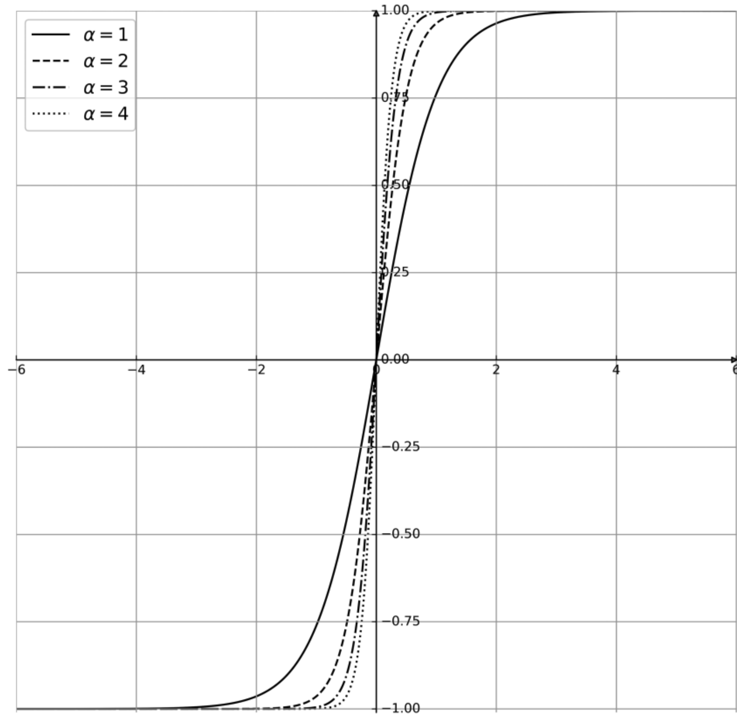


图 6-13　不同α的 Tanh 函数的图像

当α趋近于正无穷时，Tanh 函数趋近于：

$$f(x)=\begin{cases}-1, & x<0\\ 1, & x\geqslant 0\end{cases} \tag{6.19}$$

6.3.4　ReLU

线性整流单元 ReLU（rectified linear unit）是深度学习常用的激活函数，它的定义是：

$$f(x)=\mathrm{ReLU}(x)=\begin{cases}0, & x<0\\ x, & x\geqslant 0\end{cases} \tag{6.20}$$

ReLU 的导数是：

$$f'(x)=\begin{cases}0, & x<0\\ 1, & x>0\end{cases} \tag{6.21}$$

ReLU 在原点不可导。ReLU 及其导数的图像如图 6-14 所示。

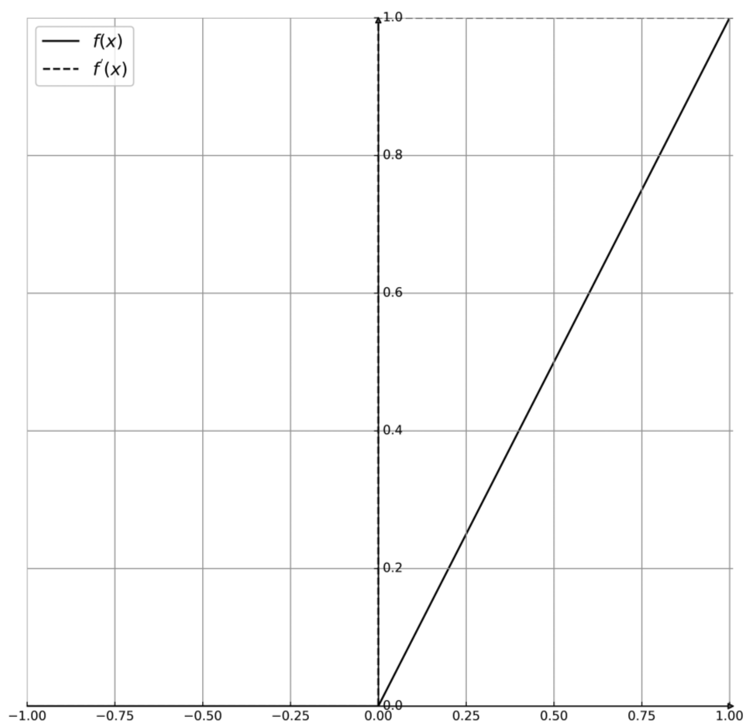


图 6-14　ReLU 及其导数的图像

ReLU 非常简单，它其实就是$\max(x,0)$。当自变量大于 0 时，ReLU 的导数为 1；当自变量小于 0 时，它的导数为 0。ReLU 在原点不可导，编程实现时，可令 ReLU 在原点的导数为 0。

相比于 Logistic 和 Tanh，ReLU 只在负侧饱和，在正侧可抬高至正无穷。ReLU 的图像是分段的两条直线，并不像 Logistic 和 Tanh 那样呈S形。但这并不影响多个神经元的合作，ReLU 神经元在抑制端为 0，其他神经元就可以在该方向上抬高输出，而互不干扰。

Logistic 和 Tanh 在两侧饱和区域的导数接近 0，这会导致神经元难以训练。ReLU 在正侧不存在饱和，一定程度上缓解了这个问题。但是当自变量为负时，ReLU 的导数为 0，一旦 ReLU 的输入为负，则神经元无法得到训练。

6.3.5　Leaky ReLU 以及 PReLU

Leaky ReLU 的负侧接近水平，但不是真的水平：

$$f(x)=\begin{cases}0.01x, & x<0\\ x, & x\geqslant 0\end{cases} \tag{6.22}$$

6

其导数是：

$$f'(x)=\begin{cases}0.01, & x<0\\ 1, & x>0\end{cases} \tag{6.23}$$

Leaky ReLU 在原点不可导，它及其导数的图像如图 6-15 所示。

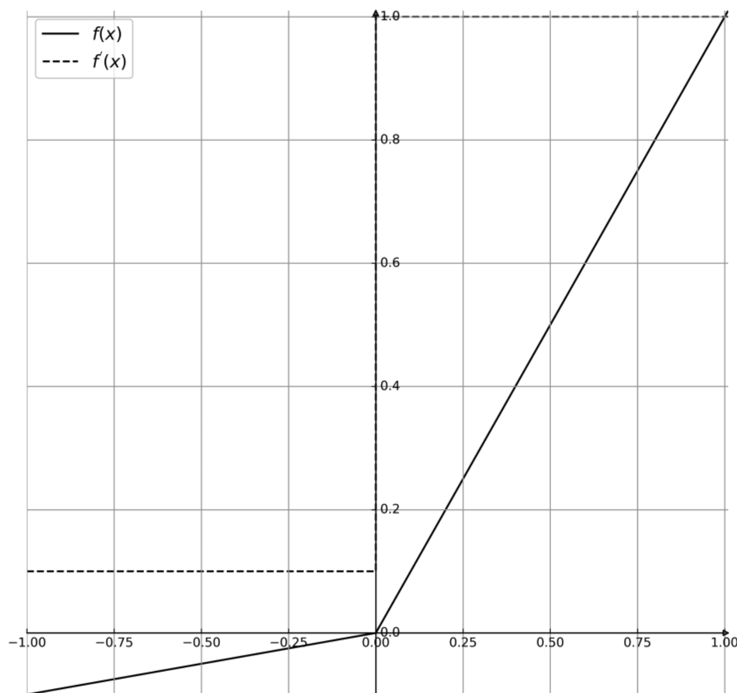


图 6-15　Leaky ReLU 及其导数的图像

Leaky ReLU 在负侧的斜率是 0.01，接近水平。它在负侧的导数虽小但不为 0，这就避免了激活函数输入为负时神经元无法得到训练的问题。若引入参数α代替 0.01，Leaky ReLU 就成了 PReLU（parameteric rectified linear unit）：

$$f(x)=\begin{cases}\alpha x, & x<0\\ x, & x\geqslant 0\end{cases} \tag{6.24}$$

PReLU 的导数是：

$$f'(x)=\begin{cases}\alpha, & x<0\\ 1, & x>0\end{cases} \tag{6.25}$$

当$\alpha=0.01$时，PReLU 就是 Leaky ReLU，当$\alpha=0$时，PReLU 就是 ReLU。

6.3.6　SoftPlus

Logistic 函数可视为“软阶跃”，它是对阶跃函数的光滑近似。ReLU 的导数是阶跃函数，那么以 Logistic 函数为导数的函数就是对 ReLU 的光滑近似：

$$f(x) = \mathrm{SoftPlus}(x) = \int \frac{1}{1+\mathrm{e}^{-x}}\,\mathrm{d}x = \int \frac{1}{1+\mathrm{e}^{x}}\,\mathrm{d}\mathrm{e}^{x} = \log(1+\mathrm{e}^{x}) \tag{6.26}$$

SoftPlus 的导数就是 Logistic 函数。SoftPlus 及其导数的图像如图 6-16 所示。

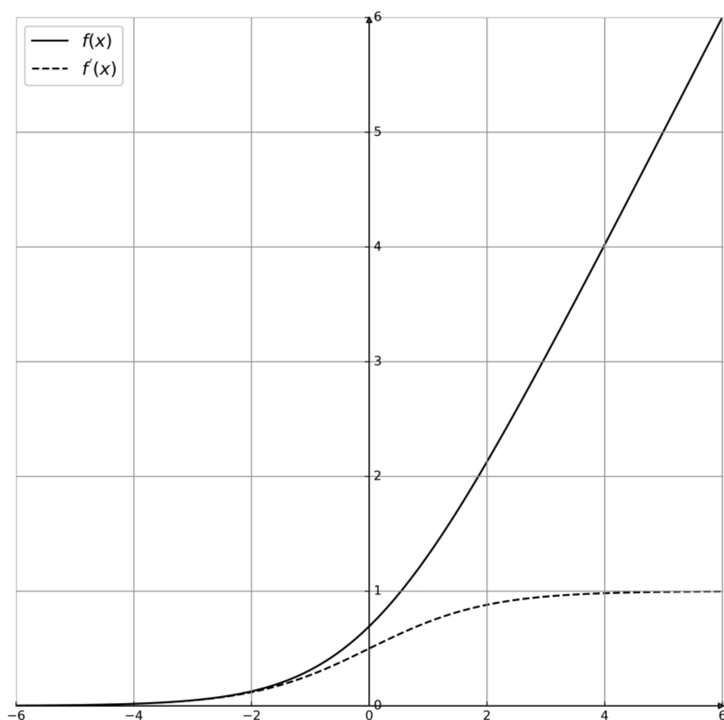


图 6-16　SoftPlus 及其导数的图像

SoftPlus 的导数在正无穷趋近于 1，在负无穷趋近于 0，其图像在正侧趋近于斜率为 1 的直线，在负侧趋近于水平线。SoftPlus 在原点的导数是 0.5。α参数可调整$\mathrm{Logistic}(x;\alpha)$函数在原点的斜率，$\alpha$越大，则$\mathrm{Logistic}(x;\alpha)$函数越接近阶跃函数。对$\mathrm{Logistic}(x;\alpha)$求积分，得到：

$$f(x;\alpha) = \int \frac{1}{1+\mathrm{e}^{-\alpha x}}\,\mathrm{d}x = \frac{1}{\alpha}\int \frac{1}{1+\mathrm{e}^{\alpha x}}\,\mathrm{d}\mathrm{e}^{\alpha x} = \frac{1}{\alpha}\log(1+\mathrm{e}^{\alpha x}) \tag{6.27}$$

不同α的$f(x;\alpha)$的图像如图 6-17 所示。

6

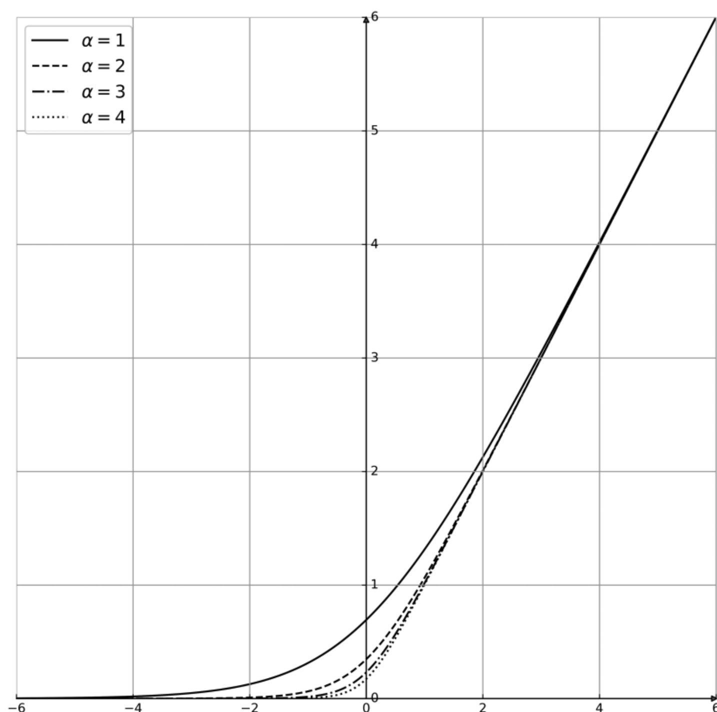


图 6-17　不同α的 SoftPlus 函数图像

随着α趋近于正无穷，SoftPlus 趋近于 ReLU。SoftPlus 是 ReLU 的光滑近似，它具备 ReLU 的良好性质，并避免了原点不可导和负侧导数为 0 的问题。但是 SoftPlus 的导数不易计算，这会对训练的效率造成影响。

6.4　多分类与 SoftMax

单个神经元只能预测二分类，而神经网络的输出层可以有任意多个神经元，可以用来预测多个类别的分类。为了能把多个神经元的输出当作多分类的概率，须要求它们在$(0,1)$区间内，且和为 1。为了达到这个要求，可对多个输出施加 SoftMax 函数。SoftMax 是一个$\mathbb{R}^n \to \mathbb{R}^n$的映射：

$$\text{SoftMax}(\boldsymbol{x}) = \sigma(\boldsymbol{x}) = \begin{pmatrix} \frac{\mathrm{e}^{x_1}}{\sum_{i=1}^{n} \mathrm{e}^{x_i}} \\ \frac{\mathrm{e}^{x_2}}{\sum_{i=1}^{n} \mathrm{e}^{x_i}} \\ \vdots \\ \frac{\mathrm{e}^{x_n}}{\sum_{i=1}^{n} \mathrm{e}^{x_i}} \end{pmatrix} \tag{6.28}$$

以下用$\sigma(\boldsymbol{x})$表示 SoftMax 函数。由式（6.28）可见，$\sigma(\boldsymbol{x})$的每一个元素都在$(0,1)$区间内，且它们的和为 1。可将$\sigma(\boldsymbol{x})$视为多分类概率分布，第 j 类的概率是：

$$p^j=\frac{\mathrm{e}^{x_j}}{\sum_{i=1}^n \mathrm{e}^{x_i}} \tag{6.29}$$

第 j 类和第 k 类的概率之比是：

$$\frac{p^j}{p^k}=\frac{\mathrm{e}^{x_j}}{\sum_{i=1}^n \mathrm{e}^{x_i}}\cdot\frac{\sum_{i=1}^n \mathrm{e}^{x_i}}{\mathrm{e}^{x_k}}=\mathrm{e}^{x_j-x_k} \tag{6.30}$$

如果只在第 j 类和第 k 类之间判断，那么结果取决于$\mathrm{e}^{x_j-x_k}$是否大于 1，也就是x_j-x_k是否大于 0。若$x_j>x_k$，则预测为第 j 类，否则预测为第 k 类。令$\boldsymbol{d}$是所有分量都是常量d的向量，有：

$$\sigma(\boldsymbol{x}-\boldsymbol{d})=\begin{pmatrix}\frac{\mathrm{e}^{x_1-d}}{\sum_{i=1}^n \mathrm{e}^{x_i-d}}\\ \frac{\mathrm{e}^{x_2-d}}{\sum_{i=1}^n \mathrm{e}^{x_i-d}}\\ \vdots\\ \frac{\mathrm{e}^{x_n-d}}{\sum_{i=1}^n \mathrm{e}^{x_i-d}}\end{pmatrix}=\begin{pmatrix}\frac{\mathrm{e}^{x_1}/\mathrm{e}^d}{\sum_{i=1}^n \mathrm{e}^{x_i}/\mathrm{e}^d}\\ \frac{\mathrm{e}^{x_2}/\mathrm{e}^d}{\sum_{i=1}^n \mathrm{e}^{x_i}/\mathrm{e}^d}\\ \vdots\\ \frac{\mathrm{e}^{x_n}/\mathrm{e}^d}{\sum_{i=1}^n \mathrm{e}^{x_i}/\mathrm{e}^d}\end{pmatrix}=\begin{pmatrix}\frac{\mathrm{e}^{x_1}}{\sum_{i=1}^n \mathrm{e}^{x_i}}\\ \frac{\mathrm{e}^{x_2}}{\sum_{i=1}^n \mathrm{e}^{x_i}}\\ \vdots\\ \frac{\mathrm{e}^{x_n}}{\sum_{i=1}^n \mathrm{e}^{x_i}}\end{pmatrix}=\sigma(\boldsymbol{x}) \tag{6.31}$$

式（6.31）表明，将每个输入x_j（$j=1,\cdots,n$）都减去常量d后，$\sigma(\boldsymbol{x})$的值不变。令 SoftMax 的输入是样本的仿射映射：

$$p(\boldsymbol{x})=\sigma(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) \tag{6.32}$$

输入向量$\boldsymbol{x}$是n维向量，即样本有n个特征，$\boldsymbol{W}$是$k\times n$矩阵，$\boldsymbol{b}$是k维偏置向量，送给 SoftMax 的输入是k维向量，可得到k个类别的概率。式（6.32）是一个k分类模型，这就是多分类逻辑回归模型。其中第i类和第j类的概率之比是：

$$\frac{p^i}{p^j}=\mathrm{e}^{(\boldsymbol{w}_{i,*})^{\mathrm{T}}\boldsymbol{x}+b_i-\left((\boldsymbol{w}_{j,*})^{\mathrm{T}}\boldsymbol{x}+b_j\right)}=\mathrm{e}^{(\boldsymbol{w}_{i,*}-\boldsymbol{w}_{j,*})^{\mathrm{T}}\boldsymbol{x}+(b_i-b_j)} \tag{6.33}$$

其中，$\boldsymbol{w}_{i,*}$是$\boldsymbol{W}$的第i行的转置，$\boldsymbol{w}_{j,*}$是$\boldsymbol{W}$的第j行的转置。式（6.33）表明第i类和第j类的分界面是一个超平面。如果类别数$k=2$，令第 1 类的概率为p，第 2 类的概率为$1-p$，则有：

$$\frac{p}{1-p}=\mathrm{e}^{(\boldsymbol{w}_{1,*}-\boldsymbol{w}_{2,*})^{\mathrm{T}}\boldsymbol{x}+(b_1-b_2)}=\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b} \tag{6.34}$$

其中，$\boldsymbol{w}=\boldsymbol{w}_{1,*}-\boldsymbol{w}_{2,*}$，$b=b_1-b_2$。对式（6.34）变形得到：

$$p=\frac{1}{1+\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}} \tag{6.35}$$

式（6.35）正是二分类逻辑回归，所以二分类逻辑回归是多分类逻辑回归的特例。多分类逻辑回归对输入向量施加仿射映射后，再施加 SoftMax 函数得到多类别的概率，如图 6-18 所示。

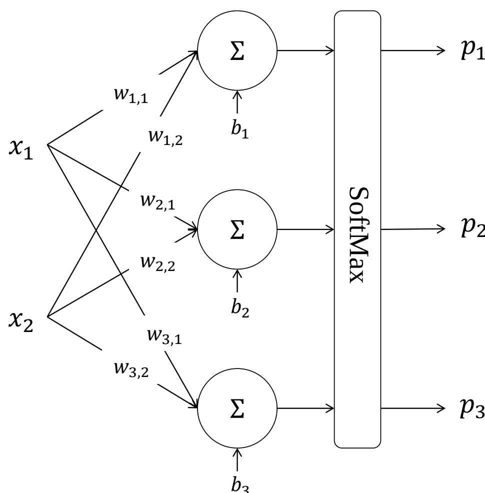


图 6-18　多分类逻辑回归的示意图

图 6-18 是一个在单神经元层之后加 SoftMax 层的简单神经网络。多隐藏层的神经网络也可以加 SoftMax 层来建模多分类问题，如图 6-19 所示。

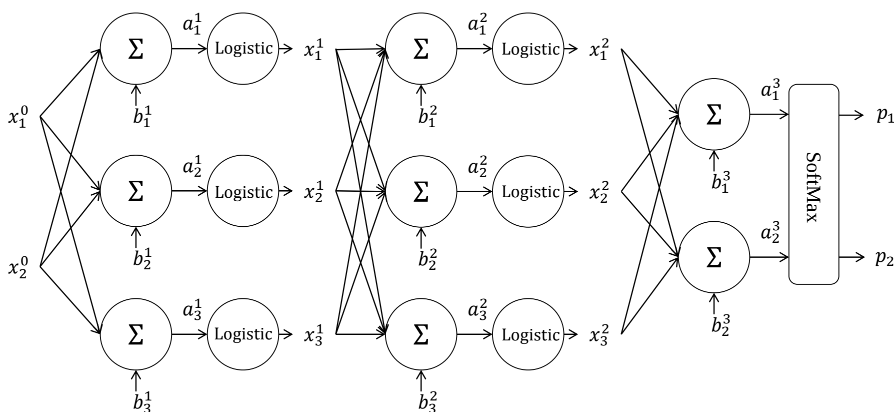


图 6-19　带 SoftMax 层的多层全连接神经网络

神经网络的行为取决于所有权值和偏置，它们是神经网络模型的参数，是待训练的对象。训练神经网络也使用梯度下降法，但偏导数的计算比较复杂。下一章将介绍计算这些偏导数的反向传播算法。

6.5　小结

单个神经元的能力是有限的，它只能产生超平面分界面，而多个神经元合作可以形成神经网络，神经网络具有非线性分类能力。本章介绍了多层全连接神经网络，并简单直观地解释了其能力的来源。

神经元对输入向量施加仿射函数后再施加激活函数，Logistic 函数是逻辑回归模型的激活函数。本章介绍了另外几种激活函数，并讨论了它们的性质。SoftMax 函数既可以视为一种激活函数，也可以视为神经网络的一层，它将神经网络的输出整理成多分类的概率，概率的大小取决于相应输入的相对大小。

阅读完本章，读者应该对多层全连接神经网络的结构和能力有了较清晰的理解，下一章中，我们将以分类问题为例讲解多层全连接神经网络的训练方法，即反向传播配合梯度下降。

第 7 章

反向传播

本章讲解如何训练多层全连接神经网络（以下简称神经网络）。训练神经网络主要使用一阶算法，例如第 3 章介绍的梯度下降法及其变体。梯度下降法需要计算损失函数对模型参数的梯度。神经网络的参数很多且关系复杂，计算它们的偏导数比逻辑回归时要复杂。反向传播算法就是计算这些偏导数的方法。

若想透彻理解反向传播算法的原理，需要掌握映射、仿射映射、雅可比矩阵以及映射求导的链式法则等知识。本章在回顾这些知识后，给出反向传播算法的详细推导和实现。本章还讨论了一些与神经网络的训练有关的话题，包括计算量、梯度消失、正则化、权值初始化和提前停止。

严格来讲，反向传播并非训练神经网络的算法，它是计算损失函数对神经网络权值和偏置的偏导数的算法。有了偏导数，也就有了梯度，更新权值和偏置用的仍是梯度下降法。反向传播是训练神经网络过程中的一个环节。需要注意的是，本章介绍的反向传播只适用于多层全连接神经网络，可以看作狭义的反向传播，它是计算图自动求导的一个特例。计算图自动求导可用于任意神经网络结构的训练，它是下一章将要介绍的内容。

7.1 映射

映射是函数的推广，函数的输出是标量，即实数，而映射的输入和输出都是向量。函数可接受多个输入，但只产生一个输出。单个神经元（如逻辑回归）就是一个函数。映射接受多个输入并产生多个输出，神经网络执行的计算就是一个映射。本节回顾关于映射的知识，包括线性映射与矩阵的关系、仿射映射、雅可比矩阵以及映射求导的链式法则。

7.1.1 仿射映射

之前讨论的都是函数（其值为标量），本节讨论的是值为向量的映射（map）。函数是映射的特例，因为标量是一维向量，下面将函数和映射统称为映射。如果对于任意向量$\boldsymbol{x}$、$\boldsymbol{y}$以及任意

实数a、b，映射$f: \mathbb{R}^n \to \mathbb{R}^m$满足：

$$f(a\boldsymbol{x}+b\boldsymbol{y}) = af(\boldsymbol{x})+bf(\boldsymbol{y}) \tag{7.1}$$

则称f是线性映射（linear map）。如果$f: \mathbb{R}^n \to \mathbb{R}^m$是线性映射，那么必然存在一个$m\times n$的矩阵$\boldsymbol{A}$，对任意$\boldsymbol{x}$满足：

$$f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} \tag{7.2}$$

下面我们来证明这个结论，任意$\boldsymbol{x}\in\mathbb{R}^n$都可以用标准基线性表出$\boldsymbol{x}=\sum_{i=1}^n x_i\boldsymbol{e}^i$，因为$f$是线性映射，所以有：

$$f(\boldsymbol{x}) = f\left(\sum_{i=1}^n x_i\boldsymbol{e}^i\right) = \sum_{i=1}^n x_i f(\boldsymbol{e}^i) = \begin{pmatrix} f(\boldsymbol{e}^1) & \cdots & f(\boldsymbol{e}^n)\end{pmatrix}\begin{pmatrix} x_1 \\ \vdots \\ x_n\end{pmatrix} = \boldsymbol{A}\boldsymbol{x} \tag{7.3}$$

$f(\boldsymbol{e}^i)$是对$\boldsymbol{e}^i$施加映射f而得到的m维向量。$m\times n$矩阵$\boldsymbol{A}=\begin{pmatrix} f(\boldsymbol{e}^1) & \cdots & f(\boldsymbol{e}^n)\end{pmatrix}$满足式（7.2）。如果线性映射的值是标量，则它的矩阵的形状是$1\times n$，是一个行向量。线性映射必将$\mathbb{R}^n$中的零向量映射到$\mathbb{R}^m$中的零向量，这是因为：

$$f(\boldsymbol{0}) = \boldsymbol{A}\boldsymbol{0} = \boldsymbol{0} \tag{7.4}$$

仿射映射（affine map）是线性映射加上一个常向量$\boldsymbol{b}$：

$$f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b} \tag{7.5}$$

如果$\boldsymbol{b}$不是零向量，则仿射映射不保持零向量。仿射映射可以看作由若干个仿射函数组成：

$$f(\boldsymbol{x}) = \begin{pmatrix} f^1(\boldsymbol{x}) \\ \vdots \\ f^m(\boldsymbol{x})\end{pmatrix} = \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b} = \begin{pmatrix} (\boldsymbol{a}_{1,*})^{\mathrm{T}}\boldsymbol{x}+b_1 \\ \vdots \\ (\boldsymbol{a}_{m,*})^{\mathrm{T}}\boldsymbol{x}+b_m\end{pmatrix} \tag{7.6}$$

f的第i分量用f^i表示，$(\boldsymbol{a}_{i,*})^{\mathrm{T}}$是矩阵$\boldsymbol{A}$的第$i$行。

7.1.2　雅可比矩阵

如果映射$f: \mathbb{R}^n \to \mathbb{R}^m$在自变量$\boldsymbol{x}$附近可以写成：

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x})+\boldsymbol{A}\boldsymbol{h}+\mathcal{R}(\boldsymbol{h}) \tag{7.7}$$

$\boldsymbol{A}$是由$\boldsymbol{x}$而定的矩阵，余项$\mathcal{R}(\boldsymbol{h})$是$m$维向量，满足：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|}=\mathbf{0} \tag{7.8}$$

则称映射f在$\boldsymbol{x}$可导。$\boldsymbol{A}$是f在$\boldsymbol{x}$的雅可比矩阵（Jacobian matrix），简称雅可比。式（7.7）的含义是$f(\boldsymbol{x}+\boldsymbol{h})$可被关于$\boldsymbol{h}$的仿射映射$f(\boldsymbol{x})+\boldsymbol{A}\boldsymbol{h}$近似，近似误差随$\boldsymbol{h}$趋近于零向量而消失，且消失得足够快——误差的每个分量都是$\|\boldsymbol{h}\|$的高阶无穷小。输出是标量的可导映射(函数)的雅可比是$1\times n$矩阵，即行向量。根据式（7.7）可知，这个雅可比行向量是该函数在$\boldsymbol{x}$的梯度的转置。

根据式（7.7），可导映射f的第i分量在$\boldsymbol{x}$附近的值$f^i(\boldsymbol{x}+\boldsymbol{h})$是：

$$f^i(\boldsymbol{x}+\boldsymbol{h})=f^i(\boldsymbol{x})+\left(\boldsymbol{a}_{i,*}\right)^{\mathrm{T}}\boldsymbol{x}+\mathcal{R}^i(\boldsymbol{h}) \tag{7.9}$$

$\left(\boldsymbol{a}_{i,*}\right)^{\mathrm{T}}$是雅可比矩阵的第$i$行，$\mathcal{R}^i(\boldsymbol{h})$是$\mathcal{R}(\boldsymbol{h})$的第$i$分量，是$\|\boldsymbol{h}\|$的高阶无穷小，所以$\boldsymbol{a}_{i,*}$是函数$f^i(\boldsymbol{x})$在$\boldsymbol{x}$的梯度：

$$\boldsymbol{a}_{i,*}=\nabla f^i(\boldsymbol{x})=\begin{pmatrix}\frac{\partial f^i(\boldsymbol{x})}{\partial x_1}\\ \vdots\\ \frac{\partial f^i(\boldsymbol{x})}{\partial x_n}\end{pmatrix} \tag{7.10}$$

所以f在$\boldsymbol{x}$的雅可比矩阵$\boldsymbol{A}$的每一行是f的每一分量在$\boldsymbol{x}$的梯度的转置：

$$\boldsymbol{A}_{m\times n}=\begin{pmatrix}\nabla f^1(\boldsymbol{x})^{\mathrm{T}}\\ \vdots\\ \nabla f^m(\boldsymbol{x})^{\mathrm{T}}\end{pmatrix}=\begin{pmatrix}\frac{\partial f^1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f^1(\boldsymbol{x})}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial f^m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f^m(\boldsymbol{x})}{\partial x_n}\end{pmatrix} \tag{7.11}$$

映射是函数的推广，雅可比是梯度的推广。

7.1.3　链式法则

我们熟悉一元复合函数$(f\oplus g)(x)=f\big(g(x)\big)$的求导链式法则：

$$\frac{\mathrm{d}(f\oplus g)(x)}{\mathrm{d}x}=f'\big(g(x)\big)g'(x) \tag{7.12}$$

可导映射的雅可比也有类似结论。若$g\colon\mathbb{R}^n\to\mathbb{R}^k$和$f\colon\mathbb{R}^k\to\mathbb{R}^m$是两个可导映射，则它们的复合$f\oplus g\colon\mathbb{R}^n\to\mathbb{R}^m$也是可导映射。如果$g$在$\boldsymbol{x}$的雅可比是$k\times n$矩阵$\boldsymbol{A}_g$，$f$在$g(\boldsymbol{x})$的雅可比是$m\times k$矩阵$\boldsymbol{A}_f$，则$f\oplus g$在$\boldsymbol{x}$的雅可比是$m\times n$矩阵$\boldsymbol{A}_f\cdot\boldsymbol{A}_g$。我们来证明这个结论，因为$g$和$f$分别在$\boldsymbol{x}$和$g(\boldsymbol{x})$可导，运用式（7.7）展开：

$$f \oplus g(\boldsymbol{x}+\boldsymbol{h}) = f\big(g(\boldsymbol{x}+\boldsymbol{h})\big) = f\left(g(\boldsymbol{x}) + \boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right) = f\big(g(\boldsymbol{x})\big) + \boldsymbol{A}_f\left(\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right)$$

$$+\mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right) = f \oplus g(\boldsymbol{x}) + \left(\boldsymbol{A}_f \cdot \boldsymbol{A}_g\right)\boldsymbol{h} + \left(\boldsymbol{A}_f \cdot \mathcal{R}(\boldsymbol{h}) + \mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right)\right) \quad (7.13)$$

要证明余项是高阶无穷小，需要证明当$\|\boldsymbol{h}\|$趋近于 0 时，$\left(\boldsymbol{A}_f \cdot \mathcal{R}(\boldsymbol{h}) + \mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right)\right)/\|\boldsymbol{h}\|$趋近于零向量。下面将两项分开来看，第一项：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\boldsymbol{A}_f \cdot \mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|} = \boldsymbol{A}_f \lim_{\|\boldsymbol{h}\|\to 0}\frac{\mathcal{R}(\boldsymbol{h})}{\|\boldsymbol{h}\|} = \boldsymbol{A}_f\boldsymbol{0} = \boldsymbol{0} \quad (7.14)$$

接着看第二项：

$$\lim_{\|\boldsymbol{h}\|\to 0}\frac{\mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\right)}{\|\boldsymbol{h}\|} = \lim_{\|\boldsymbol{h}\|\to 0}\frac{\|\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} \cdot \frac{\mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\right)}{\|\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\|} \quad (7.15)$$

当$\|\boldsymbol{h}\|$趋近于 0 时，$\boldsymbol{A}_g\boldsymbol{h}$和$\mathcal{R}(\boldsymbol{h})$都趋近于零向量，所以$\|\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\|$也趋近于 0，于是当$\|\boldsymbol{h}\|$趋近于 0 时，有$\mathcal{R}\left(\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\right)/\|\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\|$趋近于零向量。要证明整个极限趋近于零向量，还需证明系数$\|\boldsymbol{A}_g\boldsymbol{h} + \mathcal{R}(\boldsymbol{h})\|/\|\boldsymbol{h}\|$有界。因为向量之和的模不大于它们的模之和，所以有：

$$0 \leqslant \frac{\|\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} \leqslant \frac{\|\boldsymbol{A}_g\boldsymbol{h}\|+\|\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} = \left\|\boldsymbol{A}_g\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}\right\| + \frac{\|\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} \quad (7.16)$$

不论$\boldsymbol{h}$如何变化，$\boldsymbol{h}/\|\boldsymbol{h}\|$都在单位球壳上（unit sphere）。单位球壳是有界闭集——紧集（compact set），紧集上的连续函数有界，此定理本书不予证明。$\boldsymbol{A}_g\boldsymbol{h}/\|\boldsymbol{h}\|$的每个分量都是单位球壳上的连续函数，所以它们有界，于是$\|\boldsymbol{A}_g\boldsymbol{h}/\|\boldsymbol{h}\|\|$有上界$M$，所以：

$$0 \leqslant \lim_{\|\boldsymbol{h}\|\to 0}\frac{\|\boldsymbol{A}_g\boldsymbol{h}+\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|} \leqslant \lim_{\|\boldsymbol{h}\|\to 0}\left(\left\|\boldsymbol{A}_g\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}\right\| + \frac{\|\mathcal{R}(\boldsymbol{h})\|}{\|\boldsymbol{h}\|}\right) = \lim_{\|\boldsymbol{h}\|\to 0}\left\|\boldsymbol{A}_g\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}\right\| \leqslant M \quad (7.17)$$

这就证明了式（7.15）的极限是零向量。结合式（7.14）和式（7.13），说明$f \oplus g$在$\boldsymbol{x}$可导，它在$\boldsymbol{x}$的雅可比矩阵是$\boldsymbol{A}_f \cdot \boldsymbol{A}_g$，这就是映射求导的链式法则。

对于多重复合映射的求导，可以连续应用链式法则。以三重复合映射$f \oplus g \oplus h$为例，它可以看作f与$g \oplus h$的复合，它的雅可比是f与$g \oplus h$的雅可比之积，而$g \oplus h$的雅可比是g和h的雅可比之积，所以$f \oplus g \oplus h$的雅可比是f、g和h的三个雅可比之积。一元函数的雅可比是 1×1 矩阵，即标量，于是式（7.12）可看作是雅可比之积。

方向导数与梯度的关系也可以用链式法则证明，如果$\boldsymbol{d}$是单位向量，$f: \mathbb{R}^n \to \mathbb{R}$是多元函数，$f$在$\boldsymbol{x}$沿$\boldsymbol{d}$的方向导数$\nabla_{\boldsymbol{d}} f(\boldsymbol{x})$是函数$f(\boldsymbol{x} + t\boldsymbol{d})$对$t$在原点的导数。将$f(\boldsymbol{x} + t\boldsymbol{d})$拆成两个映射$f$和$g(t) = \boldsymbol{x} + t\boldsymbol{d}$的复合。$f$在$\boldsymbol{x}$的雅可比就是梯度的转置$\nabla f(\boldsymbol{x})^{\mathrm{T}}$。$g$是$\mathbb{R} \to \mathbb{R}^n$的映射，它在任意点

的雅可比都是$n \times 1$常量矩阵，即n维向量：

$$\begin{pmatrix} \frac{\mathrm{d}g^1(t)}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}g^n(t)}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} \frac{\mathrm{d}(x_1+td_1)}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}(x_n+td_n)}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} = \boldsymbol{d} \tag{7.18}$$

根据链式法则，有：

$$\nabla_{\boldsymbol{d}} f(\boldsymbol{x}) = \frac{\mathrm{d}f(\boldsymbol{x}+t\boldsymbol{d})}{\mathrm{d}t}\bigg|_{t=0} = \frac{\mathrm{d}f\big(g(t)\big)}{\mathrm{d}t}\bigg|_{t=0} = \nabla f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{d} \tag{7.19}$$

这就证明了，函数在某点沿某方向的方向导数是函数在该点的梯度向该方向的投影。

7.2　反向传播

本节介绍计算损失函数对神经网络权值和偏置的偏导数的方法——反向传播算法。具备了映射求导链式法则的知识，会发现反向传播其实就是链式法则的简单应用。神经网络的参数很多，表示起来比较繁复，这时需要点耐心。

7.2.1　网络的符号表示

首先回顾一下多层全连接神经网络的示意图和符号，如图 7-1 所示。

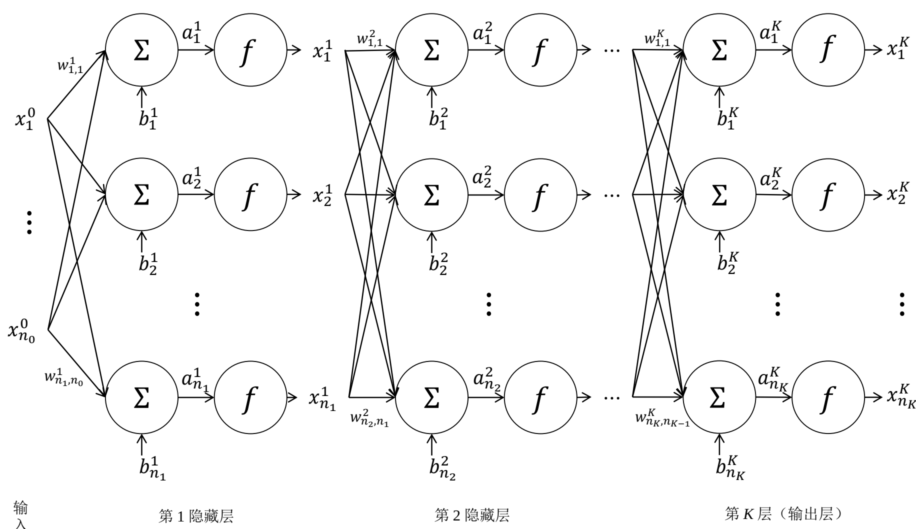


图 7-1　多层全连接神经网络的示意图和符号

我们用上标表示层的编号，输入向量是第 0 层，最接近输入的层是第 1 层，以此类推。神经网络一共有K层（不包括输入向量），输出层是第K层。第k层的神经元数量记为n_k。输入向量$\boldsymbol{x}^0$有n_0个分量，第i分量是x_i^0。第k层的第i个神经元的仿射值记为a_i^k，施加激活函数后得到的该神经元的输出记为x_i^k。因为第k层有n_k个神经元，所以该层有n_k个输出。输出层包含n_K个神经元，所以网络有n_K个输出。

第k层的输入是第$k-1$层的输出：$x_1^{k-1}, x_2^{k-1}, \cdots, x_{n_{k-1}}^{k-1}$。第$k$层的第$i$个神经元有$n_{k-1}$个权值：$w_{i,1}^k, w_{i,2}^k, \cdots, w_{i,n_{k-1}}^k$，它们构成权值向量$\boldsymbol{w}_i^k$。第$k$层的第$i$个神经元的偏置是$b_i^k$。以第$k$层的全部神经元的权值向量为行，构成$n_k \times n_{k-1}$的权值矩阵$\boldsymbol{W}^k$。以第$k$层的全部神经元的偏置为分量，构成偏置向量$\boldsymbol{b}^k$。

7.2.2　原理

用训练样本做输入向量，逐层计算直到计算出神经网络的输出，此过程称为前向传播。用网络输出和训练标签计算损失值$\mathcal{L}$。在训练样本和标签给定的情况下，损失值可视作全体权值和偏置的函数。用梯度下降法调整神经网络的权值和偏置以降低损失，这就是神经网络的训练。

梯度下降法需要计算损失值对所有权值和偏置的梯度，也就是需要计算损失值对所有权值和偏置的偏导数，如$\frac{\partial \mathcal{L}}{\partial w_{i,j}^k}$和$\frac{\partial \mathcal{L}}{\partial b_i^k}$。我们首先将一个神经元的前后连接关系画出来，如图 7-2 所示。

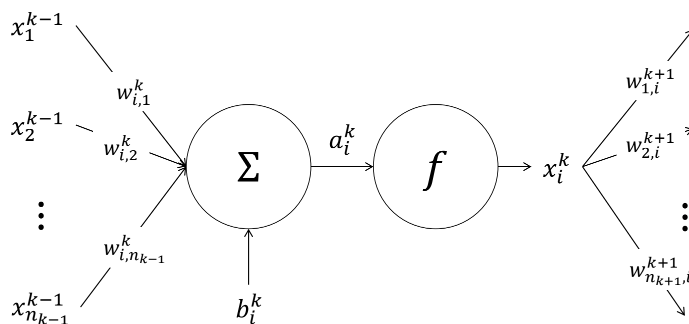


图 7-2　第k层的第i个神经元在神经网络中的前后连接关系

图 7-2 展示的是第k层的第i个神经元的前后连接关系。该神经元执行的计算是：

$$x_i^k = f\left(a_i^k\right) = f\left(\sum_{s=1}^{n_{k-1}} w_{i,s}^k x_s^{k-1} + b_i^k\right) \tag{7.20}$$

该神经元的输出x_i^k连接到下一层的各个神经元。根据链式法则，损失值对某个权值的偏导数是：

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^k} = \frac{\partial \mathcal{L}}{\partial a_i^k} \cdot \frac{\partial a_i^k}{\partial w_{i,j}^k} = x_j^{k-1} \frac{\partial \mathcal{L}}{\partial a_i^k} \tag{7.21}$$

损失值对偏置的偏导数是：

$$\frac{\partial \mathcal{L}}{\partial b_i^k} = \frac{\partial \mathcal{L}}{\partial a_i^k} \cdot \frac{\partial a_i^k}{\partial b_i^k} = \frac{\partial \mathcal{L}}{\partial a_i^k} \tag{7.22}$$

所以问题的关键是求损失值对每个神经元的仿射值的偏导数$\frac{\partial \mathcal{L}}{\partial a_i^k}$。观察第$k$层的第$i$个神经元的仿射值$a_i^k$，如图 7-3 所示。

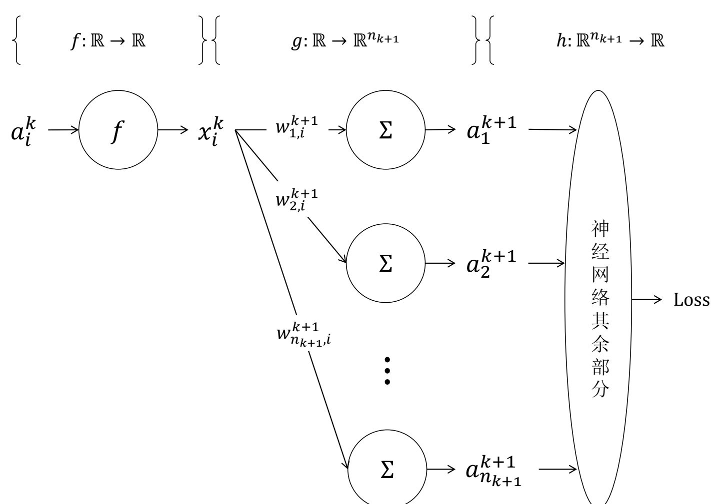


图 7-3　神经元仿射值与下一层仿射值的关系

对第k层的第i个神经元的仿射值a_i^k施加激活函数后，得到该神经元的输出x_i^k：

$$x_i^k = f\left(a_i^k\right) \tag{7.23}$$

x_i^k与同层的其他神经元的输出加权求和再加偏置，得到下一层各个神经元的仿射值。可将这步计算看作映射$g: \mathbb{R} \to \mathbb{R}^{n_{k+1}}$：

$$g\left(x_i^k\right) = \begin{pmatrix} a_1^{k+1} \\ a_2^{k+1} \\ \vdots \\ a_{n_{k+1}}^{k+1} \end{pmatrix} = \begin{pmatrix} \sum_{s=1}^{n_k} w_{1,s}^{k+1} x_s^k + b_1^{k+1} \\ \sum_{s=1}^{n_k} w_{2,s}^{k+1} x_s^k + b_2^{k+1} \\ \vdots \\ \sum_{s=1}^{n_k} w_{n_{k+1},s}^{k+1} x_s^k + b_{n_{k+1}}^{k+1} \end{pmatrix} \tag{7.24}$$

最后，映射$h: \mathbb{R}^{n_{k+1}} \to \mathbb{R}$将第$k+1$层的仿射值映射到损失值$\mathcal{L}$。若将损失值$\mathcal{L}$视作$a_i^k$的函数，则它是三个映射的复合：

$$\mathcal{L}(a_i^k) = h\left(g\left(f\left(a_i^k\right)\right)\right) \tag{7.25}$$

根据链式法则，$\mathcal{L}$对a_i^k的雅可比（导数）是这三个映射在相应位置的雅可比之积：

$$\frac{\partial \mathcal{L}}{\partial a_i^k} = \boldsymbol{A}_h \cdot \boldsymbol{A}_g \cdot \boldsymbol{A}_f \tag{7.26}$$

现在分头来看这三个雅可比。$h: \mathbb{R}^{n_{k+1}} \to \mathbb{R}$的雅可比是$1 \times n_{k+1}$矩阵：

$$\boldsymbol{A}_h = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{k+1}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{n_{k+1}}^{k+1}} \end{pmatrix} \tag{7.27}$$

第二个映射$g: \mathbb{R} \to \mathbb{R}^{n_{k+1}}$的雅可比是$n_{k+1} \times 1$矩阵，即$n_{k+1}$维向量。根据式（7.24）容易求得：

$$\boldsymbol{A}_g = \begin{pmatrix} \frac{\partial a_1^{k+1}}{\partial x_i^k} \\ \frac{\partial a_2^{k+1}}{\partial x_i^k} \\ \vdots \\ \frac{\partial a_{n_{k+1}}^{k+1}}{\partial x_i^k} \end{pmatrix} = \begin{pmatrix} w_{1,i}^{k+1} \\ w_{2,i}^{k+1} \\ \vdots \\ w_{n_{k+1},i}^{k+1} \end{pmatrix} \tag{7.28}$$

$\boldsymbol{A}_g$是由第$k+1$层的所有神经元对第k层的第i个神经元的权值$w_{1,i}^{k+1}, w_{2,i}^{k+1}, \cdots, w_{n_{k+1},i}^{k+1}$组成的向量。最后，$f: \mathbb{R} \to \mathbb{R}$的雅可比是 1×1 矩阵，即标量：

$$\boldsymbol{A}_f = f'\left(a_i^k\right) \tag{7.29}$$

将三个雅可比代回式（7.26），得到：

$$\frac{\partial \mathcal{L}}{\partial a_i^k} = \boldsymbol{A}_h \cdot \boldsymbol{A}_g \cdot \boldsymbol{A}_f = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{k+1}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{n_{k+1}}^{k+1}} \end{pmatrix} \begin{pmatrix} w_{1,i}^{k+1} \\ w_{2,i}^{k+1} \\ \vdots \\ w_{n_{k+1},i}^{k+1} \end{pmatrix} f'\left(a_i^k\right) \tag{7.30}$$

有了$\frac{\partial \mathcal{L}}{\partial a_i^k}$，就可以计算损失函数对第$k$层的第$i$个神经元的权值和偏置的偏导数了。式（7.30）就是反向传播的含义：被“传播”的是损失值对仿射值的偏导数。前一层的仿射值偏导数用后一层的仿射值偏导数计算，此为“反向”。对于每一个训练样本，前向传播计算所有中间值直到网络输出，之后向后传播仿射值的偏导数，进而计算所有权值和偏置的偏导数，这就是反向传播。计算式（7.30）的前两个矩阵（向量）之积，得到：

$$\frac{\partial \mathcal{L}}{\partial a_i^k} = f'\left(a_i^k\right) \sum_{s=1}^{n_{k+1}} w_{s,i}^{k+1} \frac{\partial \mathcal{L}}{\partial a_s^{k+1}} \tag{7.31}$$

从这个角度看，前一层神经元的仿射值偏导数等于后一层全部仿射值偏导数的加权求和，再

乘以激活函数的导数。可以将仿射值偏导数看作分配到每个神经元的局部误差，每个神经元利用自己的局部误差调整权值和偏置。局部误差从网络后部向网络前部反向传播，这是一个误差分配的过程。

7.2.3　实现

将第k层各神经元的仿射值偏导数列成行向量：

$$\Delta^k = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1^k} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{n_k}^k} \end{pmatrix} \tag{7.32}$$

根据式（7.30），有：

$$\begin{aligned} \Delta^k &= \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{k+1}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{n_{k+1}}^{k+1}} \end{pmatrix} \begin{pmatrix} w_{1,1}^{k+1} & \cdots & w_{1,n_k}^{k+1} \\ \vdots & \ddots & \vdots \\ w_{n_{k+1},1}^{k+1} & \cdots & w_{n_{k+1},n_k}^{k+1} \end{pmatrix} \begin{pmatrix} f'(a_1^k) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f'\left(a_{n_k}^k\right) \end{pmatrix} \\ &= \Delta^{k+1} \boldsymbol{W}^{k+1} \mathrm{diag}\left(f'(\boldsymbol{a}^k)\right) \end{aligned} \tag{7.33}$$

$\boldsymbol{W}^{k+1}$是第$k+1$层的权值矩阵，$\boldsymbol{a}^k$是第k层神经元的仿射值组成的向量，$f'(\boldsymbol{a}^k)$对$\boldsymbol{a}^k$的每一个分量施加激活函数的导函数，$\mathrm{diag}\left(f'(\boldsymbol{a}^k)\right)$是以$f'(\boldsymbol{a}^k)$的分量为对角线元素的对角矩阵。可以将从$\boldsymbol{a}^k$到$\boldsymbol{a}^{k+1}$的计算看作一个复合映射$g$：

$$\boldsymbol{a}^{k+1} = g(\boldsymbol{a}^k) = \boldsymbol{W}^{k+1} f(\boldsymbol{a}^k) + \boldsymbol{b}^{k+1} \tag{7.34}$$

其中，$f(\boldsymbol{a}^k)$对$\boldsymbol{a}^k$的每个分量施加激活函数。复合映射g在$\boldsymbol{a}^k$的雅可比是$\boldsymbol{W}^{k+1}\mathrm{diag}\left(f'(\boldsymbol{a}^k)\right)$，这也印证了式（7.33）。根据式（7.21），由损失值对$\boldsymbol{W}^k$每一个元素的偏导数构成的矩阵是：

$$\nabla \boldsymbol{W}^k = \begin{pmatrix} x_1^{k-1} \frac{\partial \mathcal{L}}{\partial a_1^k} & \cdots & x_{n_{k-1}}^{k-1} \frac{\partial \mathcal{L}}{\partial a_1^k} \\ \vdots & \ddots & \vdots \\ x_1^{k-1} \frac{\partial \mathcal{L}}{\partial a_{n_k}^k} & \cdots & x_{n_{k-1}}^{k-1} \frac{\partial \mathcal{L}}{\partial a_{n_k}^k} \end{pmatrix} = (\boldsymbol{x}^{k-1} \Delta^k)^{\mathrm{T}} \tag{7.35}$$

$\nabla \boldsymbol{W}^k$是$n_k \times n_{k-1}$矩阵，其每一个元素是损失值对$\boldsymbol{W}^k$对应元素的偏导数，所以在梯度下降更新时只要将$\boldsymbol{W}^k$加上$-\eta \cdot \nabla \boldsymbol{W}^k$即可，$\eta$是学习率。根据式（7.22），损失值对第$k$层的偏置向量$\boldsymbol{b}^k$的每个分量的偏导数构成的向量是$(\Delta^k)^{\mathrm{T}}$，所以更新时将$\boldsymbol{b}^k$加上$-\eta \cdot (\Delta^k)^{\mathrm{T}}$即可。还剩下一个问题就是，最后一层的仿射值偏导数向量Δ^K如何计算？

本章只讨论用多层全连接神经网络解决多分类问题。假设问题一共有n_K个类别，它同时也是输出层的神经元个数。输出层神经元不设激活函数，它们的输出就是它们的仿射值。对n_K个输出

层仿射值施加 SoftMax 函数，得到多分类概率分布：

$$\begin{pmatrix} p^1 \\ p^2 \\ \vdots \\ p^{n_K} \end{pmatrix} = \text{SoftMax}\begin{pmatrix} a_1^K \\ a_2^K \\ \vdots \\ a_{n_K}^K \end{pmatrix} = \begin{pmatrix} \frac{\mathrm{e}^{a_1^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}} \\ \frac{\mathrm{e}^{a_2^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}} \\ \vdots \\ \frac{\mathrm{e}^{a_{n_K}^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}} \end{pmatrix} \tag{7.36}$$

训练样本形如$\{\boldsymbol{x}, \boldsymbol{y}\}$，$\boldsymbol{x}$是$n_0$维向量，是神经网络的输入，标签$\boldsymbol{y}$是$n_K$维向量，其中只有一个分量是 1，其余分量都是 0。若$\boldsymbol{y}$的第i分量为 1，则表示训练样本属于第i类，这就是多分类标签的 One-Hot 编码。神经网络以训练样本$\boldsymbol{x}$为输入，输出一个多分类概率分布，就是式（7.36）。对该分布应用交叉熵损失：

$$\mathcal{L} = -\sum_{i=1}^{n_K} y_i \cdot \log p^i \tag{7.37}$$

y_i是标签向量$\boldsymbol{y}$的第i分量，当样本属于第i类时，y_i为 1 且$y_j = 0$（$j \neq i$），当样本不属于第i类时，y_i为 0，而另外某个$y_j = 1$（$j \neq i$）。计算$\mathcal{L}$对a_s^K的（偏）导数：

$$\frac{\partial \mathcal{L}}{\partial a_s^K} = -\sum_{i=1}^{n_K} \frac{y_i}{p^i} \cdot \frac{\partial p^i}{\partial a_s^K} \tag{7.38}$$

$\frac{\partial p^i}{\partial a_s^K}$的计算分两种情况：$i = s$和$i \neq s$，首先来看$i = s$时：

$$\frac{\partial p^i}{\partial a_s^K} = \frac{\partial p^s}{\partial a_s^K} = \frac{\partial}{\partial a_s^K}\left(\frac{\mathrm{e}^{a_s^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}}\right) = \frac{\mathrm{e}^{a_s^K}\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K} - \left(\mathrm{e}^{a_s^K}\right)^2}{\left(\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}\right)^2} = \frac{\mathrm{e}^{a_s^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}} - \left(\frac{\mathrm{e}^{a_s^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}}\right)^2$$

$$= p^s(1 - p^s) \tag{7.39}$$

当$i \neq s$时：

$$\frac{\partial p^i}{\partial a_s^K} = \frac{\partial}{\partial a_s^K}\left(\frac{\mathrm{e}^{a_i^K}}{\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}}\right) = -\frac{\mathrm{e}^{a_i^K}\mathrm{e}^{a_s^K}}{\left(\sum_{j=1}^{n_K} \mathrm{e}^{a_j^K}\right)^2} = -p^i p^s \tag{7.40}$$

将两种情况加代入式（7.38）：

$$\frac{\partial \mathcal{L}}{\partial a_s^K} = -\sum_{i=1}^{n_K} \frac{y_i}{p^i} \cdot \frac{\partial p^i}{\partial a_s^K} = -\frac{y_s}{p^s} p^s(1 - p^s) + \sum_{i \neq s} \frac{y_i}{p^i} p^i p^s = p^s \sum_{i=1}^{n_K} y_i - y_s = p^s - y_s \tag{7.41}$$

又一个心旷神怡的时刻：交叉熵损失函数对第s个仿射值的偏导数，是第s类的预测概率与真实概率（1 或 0）之差$p^s - y_s$，所以神经网络输出层的仿射值偏导数行向量Δ^K就是：

$$\Delta^K = \begin{pmatrix} p^1 - y_1 & \cdots & p^{n_K} - y_{n_K} \end{pmatrix} \quad (7.42)$$

令训练集是$\left\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right\}_{i=1}^{M}$，共包含$M$个样本，运用反向传播加梯度下降训练多层全连接神经网络的伪代码如下：

```
W^{k=1⋯K} ← 随机初始化
b^{k=1⋯K} ← 随机初始化
for e in 1 to epochs:
    for i in 1 to M:
        # 前向传播
        x^0 = x^(i)
        for k in 1 to K − 1:
            a^k = W^k x^{k−1} + b^k
            x^k = f(a^k)
            a^K = W^K x^{K−1} + b^K
            p = SoftMax(a^K)
        # 反向传播
        Δ^K = (p − y^(i))^T
        for k in K − 1 to 1:
            Δ^k = Δ^{k+1} W^{k+1} diag(f′(a^k))
        # 梯度下降
        for k in 1 to K:
            W^k ← W^k − η · (x^{k−1} Δ^k)^T
            b^k ← b^k − η · (Δ^k)^T
Return W^{k=1⋯K}, b^{k=1⋯K}
```

算法的外层循环共执行 epochs 轮，每轮称为一个 epoch。每轮依次对每一个训练样本执行前向传播和反向传播，并更新参数。外层循环结束后，算法返回所有权值矩阵和偏置向量。注意，交叉熵损失本应该是全部M个训练样本的交叉熵的平均：

$$\mathcal{L}(\boldsymbol{W}^{k=1\cdots K}, \boldsymbol{b}^{k=1\cdots K}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\left(\boldsymbol{W}^{k=1\cdots K}, \boldsymbol{b}^{k=1\cdots K} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \quad (7.43)$$

全体训练样本的平均交叉熵可以视为交叉熵的样本均值，它是交叉熵的期望的无偏估计，而单个训练样本的交叉熵也是交叉熵的期望的无偏估计，所以每次使用一个样本，同样是用交叉熵的期望的无偏估计来训练网络。由于每次估计只用一个样本，这就引入了较大的随机性，此方法被称为随机梯度下降（stochastic gradient descent）。

算法内层循环也可以一次取多个训练样本，例如b个，计算它们的平均交叉熵对权值和偏置的梯度，并执行梯度下降。因为平均值的梯度等于梯度的平均值，于是有：

$$\nabla\left(\frac{1}{b}\sum_{i=1}^{b}\mathcal{L}\left(\boldsymbol{W}^{k=1\cdots K},\boldsymbol{b}^{k=1\cdots K}|\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}\right)\right)=\frac{1}{b}\sum_{i=1}^{b}\nabla\mathcal{L}\left(\boldsymbol{W}^{k=1\cdots K},\boldsymbol{b}^{k=1\cdots K}|\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}\right) \quad (7.44)$$

所以可以计算这b个训练样本的梯度，再求这些梯度的平均，用平均梯度更新权值和向量。这个方法称为 Mini Batch，它相当于用b个样本估计交叉熵的期望。一批b个训练样本称为一个batch，直到全部训练样本用完，则完成了一个 epoch。如果$b=M$，那么一个 batch 包含全部训练样本，这就是普通梯度下降；如果$b=1$，即每次使用一个训练样本，这就是随机梯度下降。Mini Batch 是二者的折中。

7.3 相关问题

本节讨论与训练多层全连接神经网络有关的几个问题，包括计算量、梯度消失、正则化、权值初始化以及提前停止。从这些问题中，我们能看到深层模型独有的一些特性，深度学习主要就是应对这些新的困难和特性。

7.3.1 计算量

前向传播对神经网络各层计算仿射映射：

$$\boldsymbol{a}^k=\boldsymbol{W}^k\boldsymbol{x}^{k-1}+\boldsymbol{b}^k,\quad k=1,\cdots,K \quad (7.45)$$

如果每层的神经元数量大致为N的话，式（7.45）的时间复杂度是$O(N^2)$。整个网络的仿射映射的时间复杂度是$K\cdot O(N^2)$。反向传播阶段的计算也是同样的时间复杂度。前向传播对K个神经元层的每一层计算$f(\boldsymbol{a}^k)$：

$$f(\boldsymbol{a}^k)=\begin{pmatrix}f(a_1^k)\\ \vdots\\ f\left(a_{n_k}^k\right)\end{pmatrix},\quad k=1,\cdots,K \quad (7.46)$$

以 Logistic 激活函数为例：$f(x)=\frac{1}{1+\mathrm{e}^{-x}}$，这需要执行除法和指数运算。ReLU 在这方面优势明显，它只需要判断自变量x的符号。反向传播过程需要对K个神经元层的每一层计算$f'(\boldsymbol{a}^k)$：

$$f'(\boldsymbol{a}^k)=\begin{pmatrix}f'(a_1^k)\\ \vdots\\ f'\left(a_{n_k}^k\right)\end{pmatrix},\quad k=1,\cdots,K \quad (7.47)$$

上一章介绍激活函数时提到，Logistic 和 Tanh 的导数在已知函数值的情况下很容易计算，免去了指数和除法运算，这可以大大节省计算量。ReLU 的优势仍然明显，计算它的导数仍然只需要判断输入的符号。

7.3.2　梯度消失

反向传播阶段计算每一层的仿射值偏导数向量：

$$\Delta^k = \Delta^{k+1}\boldsymbol{W}^{k+1}\mathrm{diag}\big(f'(\boldsymbol{a}^k)\big),\quad k = 1,\cdots,K-1 \tag{7.48}$$

当输入的绝对值较大时，Logistic 和 Tanh 这种双向饱和的激活函数的导数极小，所以当神经元的仿射值绝对值较大时，损失值对它的权值和偏置的偏导数就极小，这种情况称为梯度消失。发生梯度消失时，神经元的训练将极为缓慢。

ReLU 是单向饱和的，它在正侧的导数恒为 1，避免了梯度消失问题。但 ReLU 在负侧的导数为 0，当 ReLU 神经元的仿射值为负时，在本轮迭代中它将得不到训练。Leaky ReLU 和 PReLU 规避了负侧导数为 0 的问题。SoftPlus 也具备 ReLU 的优势，但是它和它的导数的计算比 ReLU 复杂。

7.3.3　正则化

训练神经网络也可以应用$\mathcal{L}_1$或$\mathcal{L}_2$正则化。$\mathcal{L}_2$正则化的惩罚项是：

$$\mathcal{L}_2(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} \tag{7.49}$$

其中，$\boldsymbol{w}$是所有权值和偏置构成的向量，λ是正则化强度。在神经网络领域，$\mathcal{L}_2$正则化又称为权值衰减（weight decay）。正则化强度越大，则对参数绝对值的惩罚越大，优化越倾向于找到绝对值较小的参数值。绝对值较小的参数值导致绝对值较小的仿射值，对 Logistic 或 Tanh 来说，当输入靠近 0 时，它们近似于线性函数，所以强$\mathcal{L}_2$正则化使神经网络更接近线性模型。线性模型的自由度较小，强$\mathcal{L}_2$正则化降低神经网络的自由度，导致高偏置低方差，防止过拟合。

7.3.4　权值初始化

训练神经网络的第一步就是随机初始化权值和偏置，权值和偏置的初始值应该取接近 0 的随机值。上文说过，接近 0 的权值和偏置使神经网络接近线性模型，所以接近 0 的初始值使训练过程从接近线性模型开始，逐步将神经网络调整为非线性模型，于是训练过程就是一个提高模型自由度的过程，神经网络的分界面从近似超平面逐步变化为复杂的非线性分界面。

一般可用均匀分布（uniform distribution）或方差较小的正态分布来初始化权值和偏置。有一些考虑神经元输入维数和层内神经元数量的初始化方法，它们有助于避免不收敛并加速收敛，后文介绍深度学习时会讲解这类初始化方法。

7.3.5 提前停止

本章介绍的训练算法预设了 epoch 数量，也可以不预设固定的 epoch 数量，而是在每一步或几步迭代后，在独立的验证集上观察交叉熵损失和模型评价指标，如正确率、查准率、查全率和 AUC 等，当发现验证集上的损失值或评价指标达到预期或长时间没有改善，则停止训练。

有一种正则化方法叫作提前停止（early stop），如果损失值在连续若干次迭代中没有下降，则提前停止训练。具体的停止规则可以不同，参考的评价指标也可以选择，但凡是依据某种判断而提前停止训练，都属于提前停止。提前停止与$\mathcal{L}_2$正则化有联系，在全局最小点（如果有的话）$\boldsymbol{w}^*$将损失函数二阶泰勒展开：

$$\mathcal{L}(\boldsymbol{w}) \approx \mathcal{L}(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*) \tag{7.50}$$

全局最小点$\boldsymbol{w}^*$处的梯度是零向量，所以在它附近的二阶泰勒展开没有一次项。$\boldsymbol{H}$是损失函数在$\boldsymbol{w}^*$的赫森矩阵。用该二阶泰勒展开近似代表损失函数，它在$\boldsymbol{w}$的梯度是：

$$\nabla\mathcal{L}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*) \tag{7.51}$$

假设初始点是$\boldsymbol{w}^{(0)}$，它被初始化为接近零向量，令梯度下降法第t次迭代后的点是$\boldsymbol{w}^{(t)}$，有：

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta\boldsymbol{H}\left(\boldsymbol{w}^{(t-1)}-\boldsymbol{w}^*\right) \tag{7.52}$$

将式（7.52）变形：

$$\boldsymbol{w}^{(t)} - \boldsymbol{w}^* = (\boldsymbol{I}-\eta\boldsymbol{H})\left(\boldsymbol{w}^{(t-1)}-\boldsymbol{w}^*\right) \tag{7.53}$$

假设赫森矩阵$\boldsymbol{H}$是正定的，将它谱分解为$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}$，并代入式（7.53）：

$$\boldsymbol{V}\left(\boldsymbol{w}^{(t)}-\boldsymbol{w}^*\right) = (\boldsymbol{I}-\eta\boldsymbol{\Lambda})\boldsymbol{V}\left(\boldsymbol{w}^{(t-1)}-\boldsymbol{w}^*\right) \tag{7.54}$$

同理，有：

$$\boldsymbol{V}\left(\boldsymbol{w}^{(t-1)}-\boldsymbol{w}^*\right) = (\boldsymbol{I}-\eta\boldsymbol{\Lambda})\boldsymbol{V}\left(\boldsymbol{w}^{(t-2)}-\boldsymbol{w}^*\right) \tag{7.55}$$

此推导可以一直进行到$\boldsymbol{w}^{(0)}$，于是有：

$$\boldsymbol{V}\left(\boldsymbol{w}^{(t)}-\boldsymbol{w}^{*}\right)=(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t}\boldsymbol{V}\left(\boldsymbol{w}^{(0)}-\boldsymbol{w}^{*}\right) \tag{7.56}$$

因为$\boldsymbol{w}^{(0)}\approx\boldsymbol{0}$，于是有：

$$\boldsymbol{V}\boldsymbol{w}^{(t)}=(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t}\boldsymbol{V}\boldsymbol{w}^{(0)}+(\boldsymbol{I}-(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t})\boldsymbol{V}\boldsymbol{w}^{*}\approx(\boldsymbol{I}-(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t})\boldsymbol{V}\boldsymbol{w}^{*} \tag{7.57}$$

其中$(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t}$是：

$$(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t}=\begin{pmatrix}(1-\eta\lambda^{1})^{t} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & (1-\eta\lambda^{n})^{t}\end{pmatrix} \tag{7.58}$$

$\lambda^{1},\lambda^{2},\cdots,\lambda^{n}$是赫森矩阵$\boldsymbol{H}$的特征值，$\eta$是学习率。若$\eta$足够小，当迭代次数$t$趋近于无穷时，$(\boldsymbol{I}-\eta\boldsymbol{\Lambda})^{t}$趋近于零矩阵，$\boldsymbol{V}\boldsymbol{w}^{(t)}$趋近于$\boldsymbol{V}\boldsymbol{w}^{*}$，即：

$$\lim_{t\to\infty}\boldsymbol{w}^{(t)}=\lim_{t\to\infty}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{w}^{(t)}=\boldsymbol{V}^{\mathrm{T}}\lim_{t\to\infty}\boldsymbol{V}\boldsymbol{w}^{(t)}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{w}^{*}=\boldsymbol{w}^{*} \tag{7.59}$$

这就是梯度下降法逼近全局最优解的过程。如果为损失函数加上$\mathcal{L}_2$正则项：

$$\mathcal{L}(\boldsymbol{w})=\mathcal{L}(\boldsymbol{w}^{*})+\frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^{*})^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^{*})+\frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} \tag{7.60}$$

求$\mathcal{L}(\boldsymbol{w})$的驻点得到：

$$\boldsymbol{w}=(\boldsymbol{H}+\lambda\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^{*} \tag{7.61}$$

将$\boldsymbol{H}$的谱分解代入式（7.61），得到：

$$\boldsymbol{w}=(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}+\lambda\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{w}^{*}=\boldsymbol{V}^{\mathrm{T}}(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{V}\boldsymbol{w}^{*} \tag{7.62}$$

其中$(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}$是：

$$(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}=\begin{pmatrix}\frac{\lambda^{1}}{\lambda^{1}+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\lambda^{n}}{\lambda^{n}+\lambda}\end{pmatrix} \tag{7.63}$$

当正则化强度λ趋近于无穷时，$(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}$趋近于零矩阵，$\boldsymbol{w}$趋近于零向量，当$\lambda$趋近于 0 时，$(\boldsymbol{\Lambda}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}$趋近于单位矩阵，$\boldsymbol{w}$趋近于$\boldsymbol{w}^{*}$。这就是正则化强度$\lambda$对解的控制。

可见$\mathcal{L}_2$正则化强度λ的作用与梯度下降迭代次数t的作用相反，t越大或λ越小，算法解越接近全局最优解，t越小或λ越大，算法解越接近原点。提前停止也是对搜索过程的一种约束，停止越早则约束越强，算法解离原点越近。提前停止与$\mathcal{L}_2$正则化的联系如图 7-4 所示。

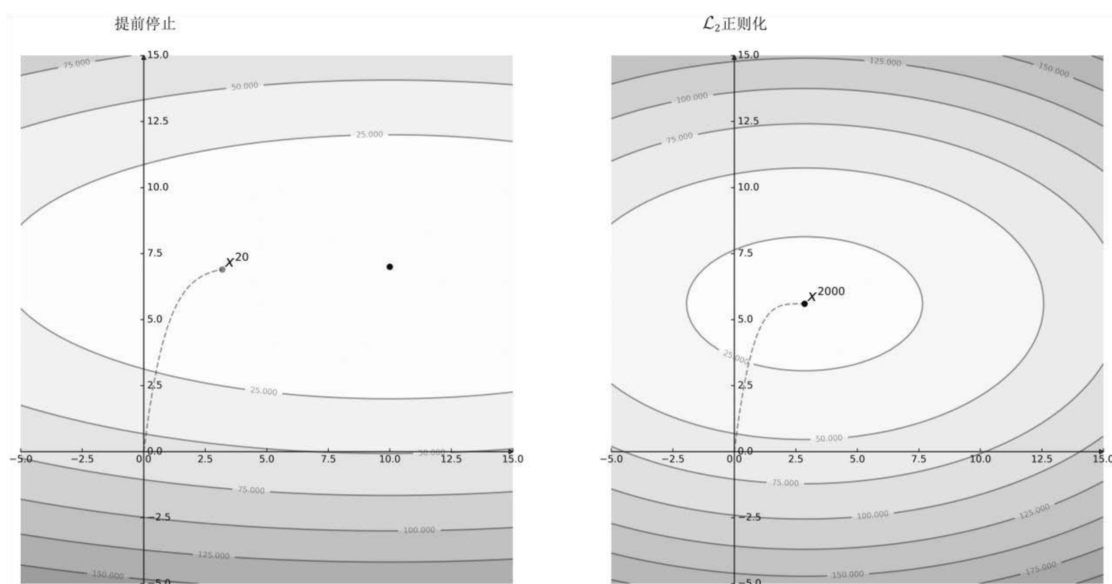


图 7-4　提前停止与$\mathcal{L}_2$正则化的联系

7.4　多层全连接神经网络的 Python 实现

本节我们实现多层全连接神经网络，训练方法是反向传播加 Adam 算法。为了清晰展现反向传播和参数更新的过程，我们没有使用第 3 章的优化器类，而是将 Adam 算法硬编码在循环体中，代码如下：


```
import numpy as np


class NeuralNetwork:

    def __init__(self, shape, activations, learning_rate = 0.005, iterations = 8000,
                 minibatch_size = 32, beta_1 = 0.9, beta_2 = 0.99):

        if not len(shape) == len(activations):
            raise Exception("激活函数名称数组必须等于层数。")

        # shape 数组是每层的神经元数量，最后一层是输出层，其神经元数量必须与问题类别数一致
        self.shape = shape
        # shape 数组的长度为神经网络的深度
        self.depth = len(self.shape)
```

```
        # 保存各层的输出值以及局部误差的数组
        self.outputs = [0] * (self.depth + 1)
        self.deltas = [0] * self.depth

        self.learning_rate = learning_rate # 学习率
        self.iterations = iterations # 迭代数量

        # 每次迭代取 minibatch_size 个样本为一批（mini batch）
        # 计算这批样本的平均交叉熵的梯度。顺序取完所有训练样本后，从头再来
        self.minibatch_size = minibatch_size

        # Adam 算法的两个超参
        self.beta_1 = beta_1
        self.beta_2 = beta_2

        # activations 数组包含各层激活函数的名称
        # 输出层直接连 SoftMax，所以最后一层的激活函数应设为恒等函数"idendity"
        self.activations = activations
        self.activation_func = []
        self.activation_func_diff = []

        for f in activations:

            f = f.lower()
            if f == "logistic":
                self.activation_func.append(self.logistic)
                self.activation_func_diff.append(self.logistic_diff)
            elif f == "identity":
                self.activation_func.append(self.identity)
                self.activation_func_diff.append(self.identity_diff)
            elif f == "relu":
                self.activation_func.append(self.relu)
                self.activation_func_diff.append(self.relu_diff)
            elif f == "tanh":
                self.activation_func.append(self.tanh)
                self.activation_func_diff.append(self.tanh_diff)
            else:
                raise Exception("不支持的激活函数：{:s}".format(f))

        # 保存各层权值矩阵和偏置向量的数组
        self.weights = [0] * self.depth
        self.biases = [0] * self.depth

        # 在未提供训练数据时，输入层的权值矩阵的尺寸未知
        # 故在这里只初始化非输入层的权值矩阵和偏置向量
```

```
        self.input_weights_initialized = False
        for idx in np.arange(1, self.depth):
            # 权值以 0 均值、0.001 标准差的正态分布初始化
            self.weights[idx] = np.mat(np.random.normal(0, 0.001,
                size=(self.shape[idx], self.shape[idx - 1])))
            # 偏置向量以 0 值初始化
            self.biases[idx] = np.mat(np.zeros((self.shape[idx], 1)))

        # 保存 Adam 算法用到的累积梯度一阶矩和二阶矩的数组
        self.weights_v = [0] * self.depth
        self.weights_s = [0] * self.depth

        self.biases_v = [0] * self.depth
        self.biases_s = [0] * self.depth
```

NeuralNetwork 类封装了我们的多层全连接神经网络实现。构造方法接受 shape 参数，它是一个整数数组，它的元素是每层的神经元数量。shape 的最后一个元素是输出层的神经元数量，它必须等于问题的类别数。shape 的长度就是网络的深度。activations 参数是与 shape 相同长度的数组，元素是相应层采用的激活函数，比如 relu、logistic。最后一层后接 SoftMax 函数，所以这一层不施加激活函数，应采用恒等函数 identity。learning_rate 参数是学习率，默认值 0.005。iterations 参数是迭代次数，默认值 8000。minibatch_size 参数是一个 Mini Batch 包含的样本数量，默认值为 32。beta_1 和 beta_2 是 Adam 算法的两个超参数，取经典默认值。构造函数初始化保存临时信息的数组，构造各层的激活函数并初始化（除输入层之外）的权值矩阵和偏置向量。

在构造函数之后，我们定义并实现 NeuralNetwork 类的成员方法，首先实现多层全连接神经网络的前向传播过程。前向传播以样本为输入，交替计算各层的仿射映射和激活函数，将网络最后一层的仿射映射值提交给 SoftMax 函数，得到多分类概率分布，代码如下：

```
def compute(self, x):
    """
    x 为 n_features * n_samples 矩阵，每列为一个样本
    本函数计算神经网络对这批样本的输出
    """
    result = x
    for idx in np.arange(0, self.depth):
        self.outputs[idx] = result
        # 对上一层的输出计算仿射值
        al = self.weights[idx] * result + self.biases[idx]
```

```
        # 对仿射值施加激活函数
        result = np.mat(self.activation_func[idx](al))

    self.outputs[self.depth] = result
    return self.softmax(result)


def predict(self, x):
    """
    x 为 n_samples * n_features 矩阵，每行为一个样本
    本函数仅仅是对 compute 函数的简单封装，供使用者执行预测之用
    """

    if not x.shape[0] or not x.shape[1]:
        raise Exception("数据为空。")

    return self.compute(x.T).T.A
```

compute 方法接受包含一批样本的矩阵，每列是一个样本。它执行前向传播，计算网络对这批样本的输出。predict 方法是 compute 方法的简单封装，它调整输入和输出的形状。

接下来，我们实现反向传播和参数更新。反向传播方法接受交叉熵损失对输出层仿射值的偏导数行向量，反向计算各层仿射值的偏导数行向量。参数更新方法利用反向传播得到的仿射值偏导数来计算各层权值矩阵和偏置向量的偏导数，并用这些偏导数更新网络参数，代码如下：

```
def bp(self, d):
    """
    反向传播
    """
    tmp = d.T

    for idx in np.arange(0, self.depth)[::-1]:
        delta = np.multiply(tmp, self.activation_func_diff[idx](self.outputs[idx + 1]).T)
        self.deltas[idx] = delta
        tmp = delta * self.weights[idx]


def update(self):
    """
    运用 Adam 算法更新权值矩阵和偏置向量
    """

    for idx in np.arange(0, self.depth):
```

```
        weights_grad = self.deltas[idx].T * self.outputs[idx].T / self.deltas[idx].shape[0]
        biases_grad = np.mean(self.deltas[idx].T, axis=1)

        # 累积权值矩阵的梯度的一阶和二阶矩
        self.weights_v[idx] = self.beta_1 * self.weights_v[idx] + (1 - self.beta_1) * weights_grad
        self.weights_s[idx] = self.beta_2 * self.weights_s[idx] + (1 - self.beta_2) *
            np.power (weights_grad, 2)

        # 累积偏置向量的梯度的一阶和二阶矩
        self.biases_v[idx] = self.beta_1 * self.biases_v[idx] + (1 - self.beta_1) * biases_grad
        self.biases_s[idx] = self.beta_2 * self.biases_s[idx] + (1 - self.beta_2) *
            np.power (biases_grad, 2)

        # 更新权值矩阵和偏置向量，在此我们省略了 Adam 中对一阶和二阶矩的微小调整
        self.weights[idx] = self.weights[idx] - self.learning_rate * self.weights_v[idx] /
            np.sqrt(self.weights_s[idx] + 1e-10)
        self.biases[idx] = self.biases[idx] - self.learning_rate * self.biases_v[idx] /
            np.sqrt(self.biases_s[idx] + 1e-10)
```

bp 方法接受输出层的误差，执行反向传播，计算损失函数对各层权值矩阵和偏置向量的偏导数。被反向传播的各层仿射值偏导数保存在 `self.deltas` 数组中。update 方法利用 `self.deltas` 数组保存的偏导数更新权值矩阵和偏置向量，该方法实现了 Adam 算法。

至此，训练过程用到的各个方法都已定义，接下来我们实现训练方法。训练方法只包含一个简单的循环体，调用之前定义的各个方法迭代地进行前向传播、反向传播以及参数更新，代码如下：

```
def train(self, x, y):

    if not x.shape[0] or not x.shape[1] or x.shape[0] != y.shape[0] or not y.shape[1]:
        raise Exception("特征矩阵与 one hot 标签矩阵的样本数不相同，或数据为空。")

    # 根据输入向量的维数初始化输入层权值矩阵和偏置向量
    if not self.input_weights_initialized:
        self.weights[0] = np.mat(np.random.normal(0, 0.001, size=(self.shape[0], x.shape[1])))
        self.biases[0] = np.mat(np.zeros((self.shape[0], 1)))
        self.input_weights_initialized = True

    start = 0
    for i in range(self.iterations):

        # 从全体训练样本中取下一批
```

```
        end = start + self.minibatch_size
        minibatch_x = x[start:end].T
        minibatch_y = y[start:end].T
        start = (start + self.minibatch_size) % x.shape[0]

        # 计算当前网络对这批样本的预测概率
        yp = self.compute(minibatch_x)

        # 计算当前模型对这批样本的正确率
        loss = np.mean(-np.sum(np.multiply(minibatch_y, np.log(yp + 1e-300)), axis=0))
        pred = np.argmax(yp, axis=0) # 取概率最大的类别为预测类别
        truth = np.argmax(minibatch_y, axis=0)
        accuracy = (pred == truth).astype(np.int).sum() / self.minibatch_size
        print("迭代：{:d}，损失值：{:.6f}，正确率：{:.2f}%".format(i, loss, accuracy * 100))

        # 误差
        d = yp - minibatch_y

        # 误差的反向传播
        self.bp(d)

        # 更新模型参数
        self.update()
```

train 方法接受特征矩阵和类别标签的 One-Hot 编码矩阵，它的主循环体共执行 iterations 次。每次迭代，按顺序从训练集中取出一批 minibatch_size 个样本，计算当前网络对这批样本输出的概率，根据标签计算误差，将误差反向传播计算梯度，用梯度更新各个权值矩阵和偏置向量。每次迭代还计算了当前网络对这批样本的平均交叉熵和预测正确率。

最后，我们实现几种常见的激活函数以及它们的导函数。它们以静态函数的形式存在于 NeuralNetwork 类中，代码如下：

```
"""
几种激活函数。在本实现中，激活函数的导函数接受的是神经元的输出
因为 logistic、tanh 和 relu 可以用输出计算导数，而且还更节省计算量
但是并非所有激活函数都可以用输出计算导数
"""
@staticmethod
def logistic(x):
    return 1.0 / (1.0 + np.power(np.e, np.where(-x > 1e2, 1e2, -x)))

@staticmethod
def logistic_diff(x):
```

```
        return np.multiply(x, (1 - x))

    @staticmethod
    def relu(x):
        return np.where(x > 0, x, 0.0)

    @staticmethod
    def relu_diff(x):
        return np.where(x > 0, 1.0, 0.0)

    @staticmethod
    def identity(x):
        return x

    @staticmethod
    def identity_diff(x):
        return np.ones(x.shape)

    @staticmethod
    def tanh(x):
        exp = 2 * np.where(x > 1e2, 1e2, x)
        return (np.power(np.e, exp) - 1) / (np.power(np.e, exp) + 1)

    @staticmethod
    def tanh_diff(x):
        return 1 - np.multiply(x, x)

    @staticmethod
    def softmax(x):
        x[x > 1e2] = 1e2
        ep = np.power(np.e, x)
        return ep / np.sum(ep, axis=0)
```

现在我们将 NeuralNetwork 类用于鸟类生态类群问题，这次我们对 6 个类别进行分类，代码如下：

```
import pandas as pd
import numpy as np
from mlp import NeuralNetwork
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler


bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)
```

```
# 先将 6 个类别的字符串名称编码成整数编号，再转化成 6 个 1/0 值的 one hot 编码
label_encoder = LabelEncoder()
one_hot_label =
OneHotEncoder(sparse=False).fit_transform(label_encoder.fit_transform(bird.type).reshape(-1, 1))

# 去掉类别的字符串名称列
bird.drop("type", axis=1, inplace=True)

# 将 10 个数值型特征列与 6 个类别标签 one hot 编码列合并起来
data = np.c_[bird.values, one_hot_label]

# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集，将特征与 one hot 编码分开
ss = StandardScaler()
train_x = ss.fit_transform(np.mat(data[:300,:-6]))
train_y = np.mat(data[:300,-6:])

# 其余样本作为测试集，将特征与 one hot 编码分开
test_x = ss.transform(np.mat(data[300:,:-6]))
test_y = np.mat(data[300:,-6:])

# 构造 2 隐藏层，每个隐藏层包含 20 个神经元的全连接神经网络，隐藏层的激活函数为 ReLU
lr = NeuralNetwork([20, 20, 6], ["relu", "relu", "identity"])

# 在训练集上训练网络
lr.train(train_x, train_y)

# 预测测试集样本的分类概率
p = lr.predict(test_x) # 模型对 6 个类别预测的概率

# 取概率最大的类别为预测类别
pred = [label_encoder.classes_[idx] for idx in np.argmax(p, axis=1)]
truth = [label_encoder.classes_[idx] for idx in np.argmax(test_y, axis=1).A1]
accuracy = accuracy_score(truth, pred)

print("正确率：{:.2f}%".format(accuracy * 100))
print(classification_report(truth, pred))
```

我们使用了 scikit-learn 库的 LabelEncoder 类和 OneHotEncoder 类。LabelEncoder 将字符串的类别标签，如 SW、R，编码成从 0 开始的整数编号。OneHotEncoder 类再将整数编号编码成 One-Hot 向量。经过这两步处理，就得到了一个“样本数 × 6”的矩阵，每一行是一个样本的类别 One-Hot

编码向量。将这个矩阵连接在 10 列特征之后，得到“样本数 × 16”的矩阵，前 10 列是特征，后 6 列是类别 One-Hot 编码。

将样本随机洗牌后，取前 300 个样本为训练集，其余样本为测试集。这里我们对特征进行了标准化处理，即对每一个特征，减去样本均值并除以样本标准差。我们用 scikit-learn 库的 `StandardScaler` 类完成此操作。注意，应该在训练集上计算样本均值和标准差，并用它们标准化训练集样本以及测试集样本。

接下来我们构造了一个 `NeuralNetwork` 对象，`shape` 参数为`[20, 20, 6]`，所以我们构造的是一个 3 层全连接神经网络，两个隐藏层各有 20 个神经元，输出层有 6 个神经元，对应问题的 6 个类别。`Activations` 参数为`["relu", "relu", "identity"]`，表示两个隐藏层的激活函数是 ReLU，输出层的激活函数是恒等函数。以训练集样本和标签调用 `train` 方法，会看到损失值下降而正确率上升的训练过程。

训练完成后，将测试集样本送给 `NeuralNetwork` 对象的 `predict` 函数，得到神经网络对测试集样本的预测概率，取概率最大的类别作为神经网络对样本的预测类别。`LabelEncoder` 对象中保存着类别整数编码与类别名称的对应关系，用这个对应关系将整数编码翻译回字符串名称，这个神经网络对 6 个类别的分类指标如表 7-1 所示。

表 7-1　6 个类别的分类指标

	查　准　率	查　全　率	f1-得分	样　本　数
P	0.91	0.77	0.83	13
R	0.86	0.92	0.89	13
SO	0.92	0.95	0.93	37
SW	0.91	0.91	0.91	35
T	0.33	1.00	0.50	1
W	0.83	0.71	0.77	14
avg / total	0.89	0.88	0.89	113

7.5　小结

在人工智能时代，反向传播算法可谓大名鼎鼎，目前它仍是最重要的神经网络训练算法。本章详细讲解了反向传播算法的原理和实现。从某个视角说，反向传播利用链式法则求各个神经元的权值和偏置的偏导数，从另一个视角说，反向传播将误差逐层分配给神经网络的每一个神经元。

通过本章的讲解，读者应该对反向传播算法有了比较透彻的理解。

但是，本章描述的反向传播对于非多层全连接的神经网络不再适用，例如深度学习中出现的那些网络。本章描述的反向传播只是计算图自动求导的一个特例。计算图自动求导是更通用的反向传播，它是通向深度学习自由王国必经之路的一把 24K 金钥匙。掌握了计算图自动求导，就能理解任何复杂网络的计算和训练，配合诸如 TensorFlow 等工具框架，可以随心所欲地构造和训练自己设计的网络结构。我们将在下一章介绍计算图和自动求导。

第 8 章

计　算　图

8

神经网络的结构并不仅限于多层全连接，在深度学习领域，存在局部连接、权值共享、跳跃连接等丰富多样的神经元连接方式，多层全连接仅仅是其中的一种。在打开更广阔的新世界的大门之前，我们首先需要掌握描述和训练任意神经网络的方法。

计算图是一个强大的工具，绝大部分神经网络都可以用计算图描述。计算图用节点表示变量，用有向边表示计算。自动求导应用链式法则求某节点对其他节点的雅可比矩阵，它从结果节点开始，沿着计算路径向前追溯，逐节点计算雅克比。将神经网络和损失函数连接成一个计算图，则它的输入、输出和参数都是节点，可利用自动求导求损失值对网络参数的雅克比，从而得到梯度。

本章首先介绍计算图，并以多层全连接神经网络和一种非全连接网络为例，展示计算图的表达能力，之后，我们介绍自动求导的原理和实现。具备了这些知识，就能理解如何构建和训练任意神经网络，为进入深度学习领域做好准备。

8.1　计算图模型

我们需要一种灵活通用的方法描述各种神经网络，计算图就是一种合适的工具。本节首先介绍计算图的基本概念，之后阐述如何用计算图描述多层全连接神经网络以及一个简单的卷积神经网络。

8.1.1　简介

计算图（computational graph）是一种有向无环图（directed acyclic graph，DAG）。计算图用节点表示变量，用有向边（directed edge）表示计算。有向边的目的节点称为子节点，源节点称为父节点，计算图定义如何用父节点计算子节点，如图 8-1 所示。

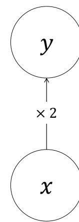


图 8-1　计算图的节点和有向边

图 8-1 中的计算图描述了计算：

$$y = 2x \tag{8.1}$$

一个子节点可以有两个父节点，表示该子节点由两个父节点计算而得，如图 8-2 所示。

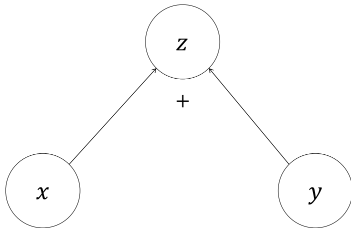


图 8-2　有两个父节点的子节点

图 8-2 中的计算图描述的计算是：

$$z = x + y \tag{8.2}$$

一个子节点也可以有两个以上的父节点，但是这种情况可以通过添加中间节点而转化成每个子节点只有两个父节点的情况。图 8-3 中的两个计算图是等价的。

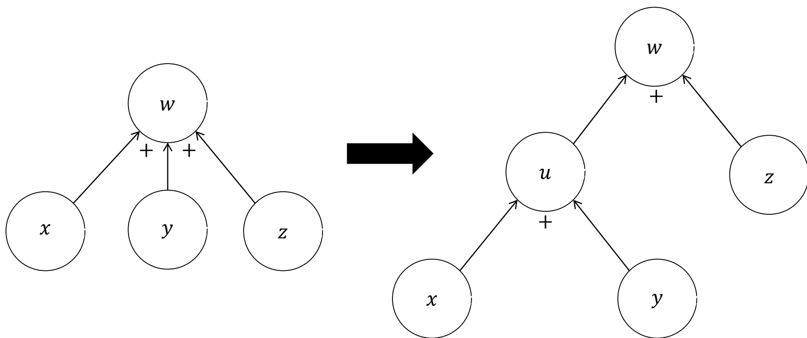


图 8-3　两个以上父节点可以转化成两个父节点

图 8-3 中的两个计算图描述的计算都是：

$$w = x + y + z \tag{8.3}$$

所以本书中，每个子节点至多有两个父节点。用上述简单的组件可以表达复杂的计算，例如图 8-4 所示。

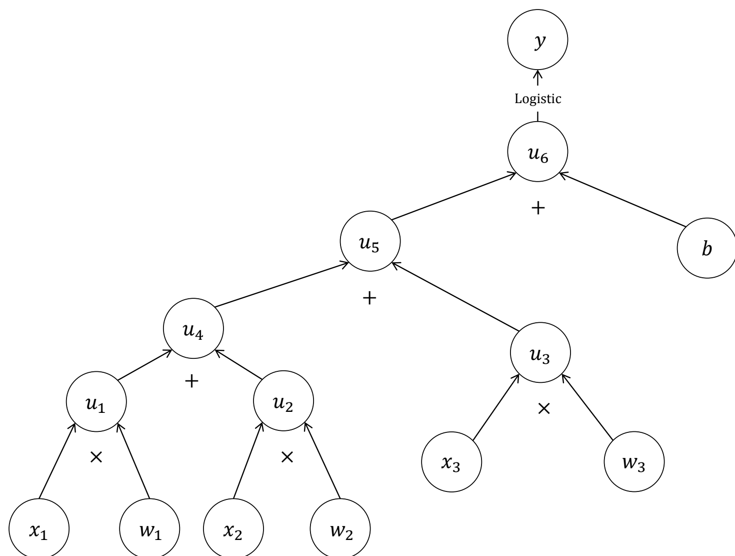


图 8-4 逻辑回归的计算图

图 8-4 中的计算图表示逻辑回归模型：

$$y = \frac{1}{1+\mathrm{e}^{-(w_1x_1+w_2x_2+w_3x_3+b)}} \tag{8.4}$$

同一个计算可以用不同的计算图描述。计算式可以代数变形这自不必说，即便是同一个式子也可以用不同的计算图描述，这取决于表示计算的“粒度”。例如图 8-4 中，从u_6节点得到y节点的计算是 Logistic 函数，但也可以将 Logistic 函数拆解成更基础的计算，如图 8-5 所示。

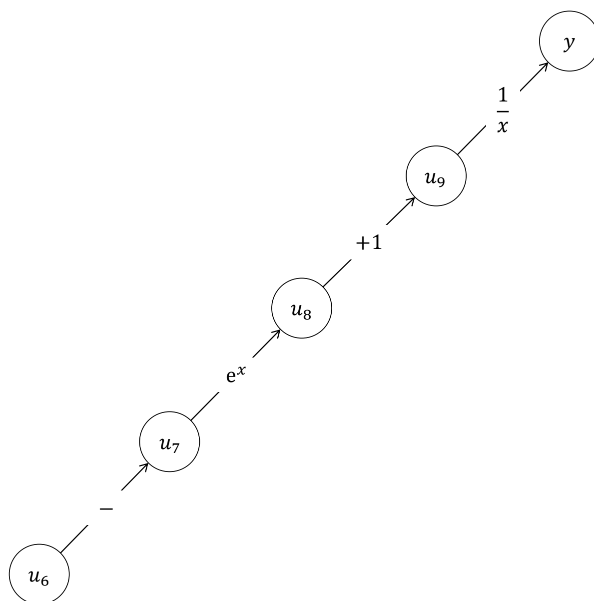


图 8-5　将 Logistic 函数拆解成更基础的计算

图 8-5 中的计算图只包含取反、指数、增 1 和取倒数这四种基础运算，它表示的是 Logistic 函数。用更基础的运算构建计算图，会使计算图的规模更大。以上介绍的计算图的节点都是标量，节点也可以是向量、矩阵乃至张量（tensor）。可将矩阵或张量的元素重新排列为向量，例如矩阵：

$$\boldsymbol{X}=\begin{pmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{pmatrix} \tag{8.5}$$

将$\boldsymbol{X}$的元素重新排列，可以得到向量：

$$\boldsymbol{x}=\begin{pmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \\ \vdots \\ x_{m,1} \\ \vdots \\ x_{m,n} \end{pmatrix} \tag{8.6}$$

向量$\boldsymbol{x}$是将$\boldsymbol{X}$的各行排成一列，它的维数是mn。对矩阵或张量的计算无非是对其元素的计算，所以它们都可以转化成对向量的计算。若计算的结果是矩阵或张量，也可以将其排列成向量。所以本书讨论的计算图的节点都是向量或标量。使用向量节点，逻辑回归模型可以表示为图 8-6 中的计算图。

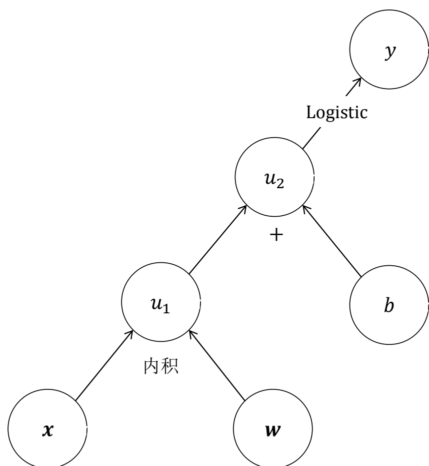


图 8-6 节点为向量的逻辑回归模型计算图

图 8-6 中的$\boldsymbol{x}$和$\boldsymbol{w}$是向量，用它们得到标量节点u_1的运算是内积。若计算图有多个输入节点，即该计算图接受多个输入向量，可将这些向量连在一起视为一个输入向量。整个计算图本质上是一个映射：由输入向量得到输出向量。每个节点和它的父节点构成计算图的一个子计算图，它也是一个映射。

如果计算图有多个输入节点，可将其中一部分输入节点视为变量，将其他输入节点视为常量。例如图 8-6 中的逻辑回归计算图，预测时将$\boldsymbol{w}$和b视为常量，将$\boldsymbol{x}$视为变量；训练时则将$\boldsymbol{x}$视为常量，将$\boldsymbol{w}$和b视为变量。

8.1.2 多层全连接神经网络的计算图

上一章出现的多层全连接神经网络示意图就是一个计算图，它的节点都是标量，表示计算的粒度较细。现在我们可以用向量节点更简洁地表示多层全连接神经网络，如图 8-7 所示。

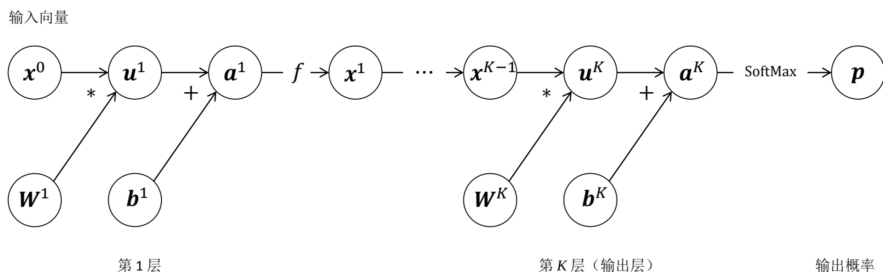


图 8-7 用向量节点计算图表示多层全连接神经网络

$\boldsymbol{W}^k$是权值矩阵，$\boldsymbol{b}^k$是偏置向量。将输入$\boldsymbol{x}^{k-1}$与权值矩阵$\boldsymbol{W}^k$相乘，得到向量$\boldsymbol{u}^k$，将$\boldsymbol{u}^k$与$\boldsymbol{b}^k$相加得到仿射值向量$\boldsymbol{a}^k$，对$\boldsymbol{a}^k$的每个分量施加激活函数f，得到该神经元层的输出$\boldsymbol{x}^k$。输出层不对仿射值向量$\boldsymbol{a}^K$施加激活函数，而是施加 SoftMax 函数，得到多分类概率向量$\boldsymbol{p}$。

8.1.3　其他神经网络结构的计算图

计算图可以灵活地表达各种复杂的神经网络，本节举一个例子，请看图 8-8 所示的神经网络。

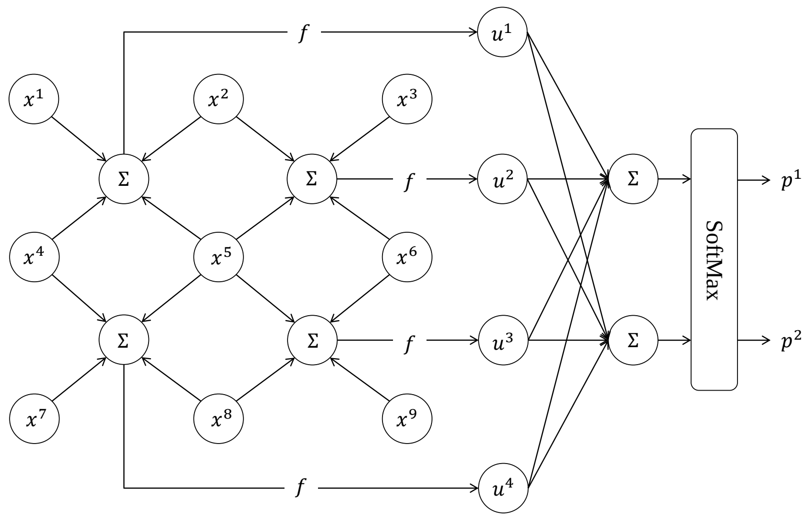


图 8-8　非全连接结构的神经网络

这个神经网络的输入是$x_1, x_2, \cdots, x_9$。将输入排成 3×3 的阵列，该阵列包含 4 个 2×2 的子阵列，将每个子阵列的输入加权求和再加偏置得到仿射值。这 4 个仿射单元使用同一套权值和偏置（图中没有画出）。对 4 个仿射值施加激活函数f，然后连接到两个仿射单元。对这两个仿射单元的输出施加 SoftMax 函数，得到两个概率p^1和p^2。后文会知道，这个神经网络就是一个卷积层加一个全连接层的卷积神经网络。用矩阵运算来表示该神经网络的计算：

$$(p^1, p^2) = \text{SoftMax}\left(f\left(\boldsymbol{W}_{1\times4}^1 \cdot \sum_{i=1}^4 \boldsymbol{S}_{4\times9}^i \boldsymbol{x}_{9\times1} \boldsymbol{C}_{1\times4}^i + b \cdot (1,1,1,1)\right) \cdot \boldsymbol{W}_{4\times2}^2 + \boldsymbol{b}_{1\times2}\right) \quad (8.7)$$

式（8.7）中的$\boldsymbol{W}_{4\times2}^2$是全连接层的权值矩阵，每一列是一个神经元的权值向量。$\boldsymbol{b}_{1\times2}$是全连接层的偏置行向量，每个分量是一个神经元的偏置。b是卷积层的 4 个仿射单元共同的偏置，它乘上矩阵(1,1,1,1)得到(b, b, b, b)。$\boldsymbol{W}_{1\times4}^1$是卷积层的 4 个仿射单元共同的权值矩阵$(w_1^1, w_2^1, w_3^1, w_4^1)$。$\boldsymbol{x}_{9\times1}$是由 9 个输入构成的列向量。$\boldsymbol{S}_{4\times9}^i$是 4 个选择矩阵，它们负责为卷积层的每个仿射单元选择适当的输入，例如：

$$S_{4\times 9}^{1}=\begin{pmatrix}1&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0\\0&0&0&0&1&0&0&0&0\end{pmatrix} \tag{8.8}$$

行向量$\boldsymbol{C}_{1\times 4}^{i}$的第$i$分量是 1，其余分量是 0，例如：

$$\boldsymbol{C}_{1\times 4}^{1}=(1,0,0,0) \tag{8.9}$$

于是容易验证：

$$\boldsymbol{S}_{4\times 9}^{1}\boldsymbol{x}\boldsymbol{C}_{1\times 4}^{1}=\begin{pmatrix}x_1&0&0&0\\x_2&0&0&0\\x_4&0&0&0\\x_5&0&0&0\end{pmatrix} \tag{8.10}$$

所以有：

$$\sum_{i=1}^{4}\boldsymbol{S}_{4\times 9}^{i}\boldsymbol{x}_{9\times 1}\boldsymbol{C}_{1\times 4}^{i}=\begin{pmatrix}x_1&x_2&x_4&x_5\\x_2&x_3&x_5&x_6\\x_4&x_5&x_7&x_8\\x_5&x_6&x_8&x_9\end{pmatrix} \tag{8.11}$$

所以$\boldsymbol{W}_{1\times 4}^{1}\cdot\sum_{i=1}^{4}\boldsymbol{S}_{4\times 9}^{i}\boldsymbol{x}_{9\times 1}\boldsymbol{C}_{1\times 4}^{i}+b\cdot(1,1,1,1)$就是卷积层的 4 个仿射单元的输出。之后的计算很容易理解。式（8.7）的计算图如图 8-9 所示。

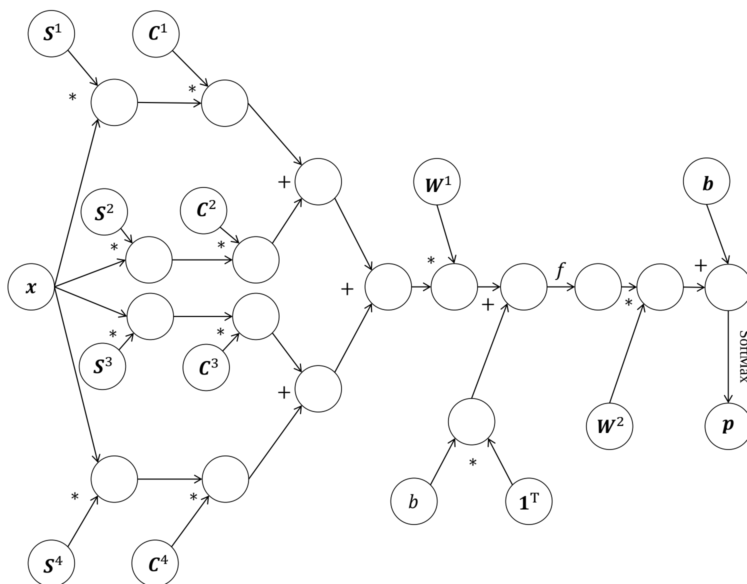


图 8-9　卷积神经网络的计算图

由于我们将计算限制为矩阵/向量的加法和乘法以及激活函数，而没有其他类型的计算，故该计算图稍复杂。如果加入更多的计算类型，则可以更简洁地表示这个卷积神经网络，但是这个例子说明计算图足以表达复杂的神经网络。

8.2　自动求导

对于计算图中的任意节点$\boldsymbol{x}$和$\boldsymbol{y}$，如果存在一条从$\boldsymbol{x}$到$\boldsymbol{y}$的有向路径，其他节点都看作常量，则$\boldsymbol{y}$可以视为$\boldsymbol{x}$的映射，如图 8-10 所示。

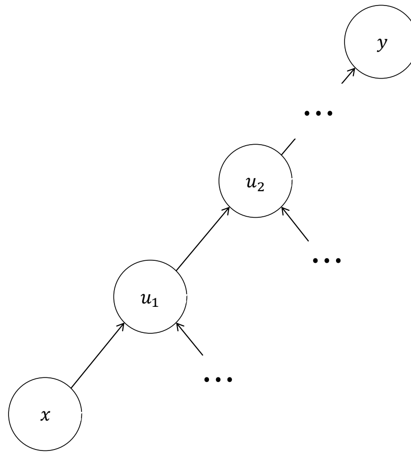


图 8-10　计算图中的一条有向路径表示一个映射

图 8-10 省略了计算图的其他部分。由$\boldsymbol{x}$计算$\boldsymbol{y}$是一个多重复合映射，根据链式法则，$\boldsymbol{y}$对$\boldsymbol{x}$的雅可比矩阵是：

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{u}^k} \cdot \frac{\partial \boldsymbol{u}^k}{\partial \boldsymbol{u}^{k-1}} \cdots \frac{\partial \boldsymbol{u}^2}{\partial \boldsymbol{u}^1} \cdot \frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{x}} \tag{8.12}$$

式(8.12)中的偏导数都是雅可比矩阵。如果要计算$\boldsymbol{y}$对$\boldsymbol{x}$在$\boldsymbol{x}^*$的雅可比矩阵，则需要计算$\boldsymbol{u}^1$对$\boldsymbol{x}$在$\boldsymbol{x}^*$的雅可比矩阵$\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{x}}$，$\boldsymbol{u}^2$对$\boldsymbol{u}^1$在$\boldsymbol{u}^1(\boldsymbol{x}^*)$的雅可比矩阵$\frac{\partial \boldsymbol{u}^2}{\partial \boldsymbol{u}^1}$，以此类推。如果结果$y$是标量，则雅克比矩阵$\frac{\partial y}{\partial \boldsymbol{x}}$是一个行向量，是$y$对$\boldsymbol{x}$在$\boldsymbol{x}^*$的梯度的转置。

如果一个节点有多个子节点，如图 8-11 所示，节点$\boldsymbol{x}$经过两个子节点$\boldsymbol{u}^1$和$\boldsymbol{u}^2$计算出最终结果。假设现在$\frac{\partial y}{\partial \boldsymbol{u}^1}$和$\frac{\partial y}{\partial \boldsymbol{u}^2}$已知，该如何计算$\frac{\partial y}{\partial \boldsymbol{x}}$呢？如果$\boldsymbol{u}^1$是$m$维向量，$\boldsymbol{u}^2$是$k$维向量，可将它们连在一起构成$(m+k)$维向量$\boldsymbol{u}$：

$$\boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}^1 \\ \boldsymbol{u}^2 \end{pmatrix} \tag{8.13}$$

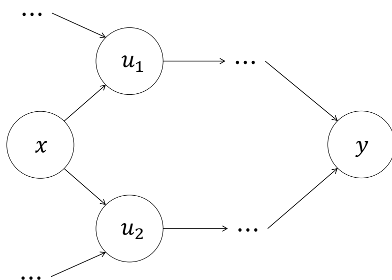


图 8-11 一个节点有多个子节点

y对$\boldsymbol{u}$的雅可比是$1\times(m+k)$矩阵：

$$\frac{\partial y}{\partial \boldsymbol{u}}=\begin{pmatrix}\frac{\partial y}{\partial u_1^1} & \cdots & \frac{\partial y}{\partial u_m^1} & \frac{\partial y}{\partial u_1^2} & \cdots & \frac{\partial y}{\partial u_k^2}\end{pmatrix}=\begin{pmatrix}\frac{\partial y}{\partial \boldsymbol{u}^1} & \frac{\partial y}{\partial \boldsymbol{u}^2}\end{pmatrix} \tag{8.14}$$

$\boldsymbol{u}$对$\boldsymbol{x}$的雅可比是$(m+k)\times n$矩阵：

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}=\begin{pmatrix}\frac{\partial u_1^1}{\partial x_1} & \cdots & \frac{\partial u_1^1}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial u_m^1}{\partial x_1} & \cdots & \frac{\partial u_m^1}{\partial x_n}\\ \frac{\partial u_1^2}{\partial x_1} & \cdots & \frac{\partial u_1^2}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial u_k^2}{\partial x_1} & \cdots & \frac{\partial u_k^2}{\partial x_n}\end{pmatrix}=\begin{pmatrix}\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{x}}\\ \frac{\partial \boldsymbol{u}^2}{\partial \boldsymbol{x}}\end{pmatrix} \tag{8.15}$$

根据链式法则，y对$\boldsymbol{x}$的雅可比是$1\times n$矩阵：

$$\frac{\partial y}{\partial \boldsymbol{x}}=\frac{\partial y}{\partial \boldsymbol{u}}\cdot\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}=\begin{pmatrix}\frac{\partial y}{\partial \boldsymbol{u}^1} & \frac{\partial y}{\partial \boldsymbol{u}^2}\end{pmatrix}\begin{pmatrix}\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{x}}\\ \frac{\partial \boldsymbol{u}^2}{\partial \boldsymbol{x}}\end{pmatrix}=\frac{\partial y}{\partial \boldsymbol{u}^1}\cdot\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{x}}+\frac{\partial y}{\partial \boldsymbol{u}^2}\cdot\frac{\partial \boldsymbol{u}^2}{\partial \boldsymbol{x}} \tag{8.16}$$

如果$\boldsymbol{x}$有两个以上子节点，同样可以证明：

$$\frac{\partial y}{\partial \boldsymbol{x}}=\sum_{i=1}^{s}\frac{\partial y}{\partial \boldsymbol{u}^i}\cdot\frac{\partial \boldsymbol{u}^i}{\partial \boldsymbol{x}} \tag{8.17}$$

这对于自动求导非常有利：如果一个节点有多个子节点，将结果节点对这些子节点的雅可比与这些子节点对该节点的雅可比相乘再求和，就得到了结果节点对该节点的雅克比。有时需要计算y对多个节点的梯度，如图 8-12 所示。

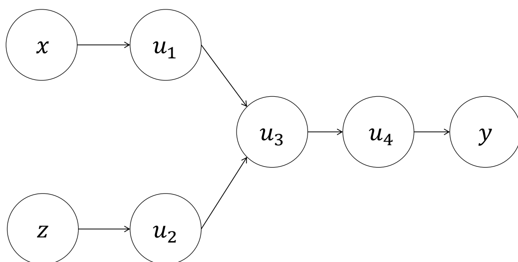


图 8-12 计算节点对多个上游节点的梯度

假设图 8-12 中的$\boldsymbol{x}$是n维向量，$\boldsymbol{z}$是m维向量，y是标量，可将该计算图视作映射$f: \mathbb{R}^{n+m} \rightarrow \mathbb{R}$。可以两次应用式（8.12）分别计算$f$对$\boldsymbol{x}$和$\boldsymbol{z}$的梯度，但是注意下式：

$$\frac{\partial y}{\partial \boldsymbol{u}^3} = \frac{\partial y}{\partial \boldsymbol{u}^4} \cdot \frac{\partial \boldsymbol{u}^4}{\partial \boldsymbol{u}^3} \tag{8.18}$$

式（8.18）会被计算两次，如果将其结果保存起来，则可以节省计算量。这就是自动求导的核心所在：保存结果节点对计算路径上各个节点的雅可比，并用它们计算结果节点对更上游节点的雅可比。中间节点的雅克比就是上一章中仿射值偏导数的推广，是被“反向传播”的对象。计算图自动求导是广义的反向传播。

8.3 自动求导的实现

本节讨论计算图自动求导的实现，我们以面向对象式的伪代码来描述该实现。节点是对象，它包含两个属性：value 和 jacobi。value 包含本节点的值，如果本节点尚未被计算，则 value 是 NULL。jacobi 包含结果节点对本节点的雅可比，如果雅可比尚未被计算，则 jacobi 为 NULL。节点有如下方法。

- evaluate()计算节点的值，如果有父节点的值尚未计算，则抛出异常；
- get_children()返回所有子节点，若无子节点则返回空集；
- get_parents()返回所有父节点，若无父节点则返回空集；
- get_jacobi(v)接受一个父节点，计算本节点对这个父节点的雅可比。注意本方法与 jacobi 属性的区别，jacobi 是结果节点对本节点的雅可比，get_jacobi(v)是计算并返回本节点对某个父节点的雅可比。

若要计算某个节点的值，则它的所有父节点必须先被计算。信息沿着计算图路径从前向后传播，这就相当于神经网络的前向传播。以下 forward_propagation 函数实现了计算某节点值的前向传播过程。

```
function forward_propagation(v):
    for p in v.get_parents():
        if p.value is NULL:
            forward_propagation(p)
    v.evaluate()
return v.value
```

节点的 evaluate()执行的计算可以是标量计算，矩阵/向量计算或者其他更复杂的计算，忽略各种计算的差异，将它们的时间复杂度都视为$O(1)$，若计算图有n个节点，则它的时间复杂度是$O(n)$。

若要计算某个节点对它的某个上游节点的雅克比，则沿着计算图路径从后向前，逐节点计算结果节点对它们的雅克比。在所有子节点的雅克比计算完成后，父节点的雅克比可用式（8.17）计算。中间节点的雅克比可能会被使用多次，将它们保存在对象属性（jacobi）中，可避免重复计算。以下 back_propagation 函数计算节点y对某个上游节点v的雅可比。

```
function back_propagation(y,v):
    if v.jacobi is NULL:
        if v == y:
            v.jacobi = I
        else:
            v.jacobi = O  # O为零矩阵
            for c in v.get_children():
                v.jacobi += back_propagation(y,c) * c.get_jacobi(v)
return v.jacobi
```

get_jacobi 对计算路径上的每条边执行一次，n个节点的计算图最多有$C_n^2=\frac{n(n-1)}{2!}$条边，如果认为所有 get_jacobi 的时间复杂度都是$O(1)$，则自动求导的时间复杂度是$O(n^2)$。试想如果粗暴地直接应用链式法则，则中间节点的雅可比有可能被重复计算多次。反向传播的本质是以空间换时间，将中间节点的雅可比保存起来，重复使用。父节点的雅可比根据其子节点的雅可比计算，信息沿着计算路径向前传播，这就是反向传播的含义。

将自动求导应用于图 8-7 所示的多层全连接神经网络，计算交叉熵$\mathcal{L}$对各层权值和偏置向量的雅克比。首先以第一层权值矩阵$\boldsymbol{W}^1$为例，从$\boldsymbol{W}^1$到$\mathcal{L}$的计算路径非常简单：

$$\boldsymbol{W}^1 \to \boldsymbol{u}^1 \to \boldsymbol{a}^1 \to \boldsymbol{x}^1 \to \cdots \to \boldsymbol{x}^{K-1} \to \boldsymbol{u}^K \to \boldsymbol{a}^K \to \boldsymbol{p} \to \mathcal{L} \tag{8.19}$$

最后一步是以神经网络的输出概率$\boldsymbol{p}$和训练样本的标签 One-Hot 编码向量$\boldsymbol{y}$计算交叉熵$\mathcal{L}$。这是一个稍复杂的计算，但仍可以将其视作计算图的一步。在第 7 章我们知道，交叉熵$\mathcal{L}$对 SoftMax 的输入向量$\boldsymbol{a}^K$的雅可比是：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^K}=\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}}\cdot\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{a}^K}=(\boldsymbol{p}-\boldsymbol{y})^{\mathrm{T}} \tag{8.20}$$

第k层的$\boldsymbol{a}^k=\boldsymbol{u}^k+\boldsymbol{b}^k$，所以：

$$\frac{\partial \boldsymbol{a}^k}{\partial \boldsymbol{u}^k}=\boldsymbol{I} \tag{8.21}$$

第k层的$\boldsymbol{u}^k=\boldsymbol{W}^k\boldsymbol{x}^{k-1}$。$\boldsymbol{W}^k$是第$k$层的权值矩阵，有：

$$\frac{\partial \boldsymbol{u}^k}{\partial \boldsymbol{x}^{k-1}}=\boldsymbol{W}^k \tag{8.22}$$

第$k-1$层的输出$\boldsymbol{x}^{k-1}=f(\boldsymbol{a}^{k-1})$，所以：

$$\frac{\partial \boldsymbol{x}^{k-1}}{\partial \boldsymbol{a}^{k-1}}=\begin{pmatrix} f'(a_1^{k-1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f'\left(a_{n_{k-1}}^{k-1}\right) \end{pmatrix}=\mathrm{diag}\left(f'(\boldsymbol{a}^{k-1})\right) \tag{8.23}$$

$\boldsymbol{u}^1=\boldsymbol{W}^1\boldsymbol{x}^0$，输入$\boldsymbol{x}^0$是$n_0$维向量，权值矩阵$\boldsymbol{W}^1$是$n_1\times n_0$矩阵。令向量$\boldsymbol{w}^1$是将$\boldsymbol{W}^1$的各行连接起来形成的向量，它是$n_1\cdot n_0$维向量。$\boldsymbol{u}^1$对$\boldsymbol{w}^1$的雅可比是$n_1\times(n_1\cdot n_0)$矩阵：

$$\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{w}^1}=\begin{pmatrix} x_1^0 & \cdots & x_{n_0}^0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & x_1^0 & \cdots & x_{n_0}^0 \end{pmatrix}_{n_1\times(n_1\cdot n_0)} \tag{8.24}$$

将上述一系列雅可比连乘起来就得到交叉熵$\mathcal{L}$对$\boldsymbol{w}^1$的雅可比矩阵：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^1}=(\boldsymbol{p}-\boldsymbol{y})^{\mathrm{T}}\left(\prod_{k=K}^{2}\boldsymbol{W}^k\mathrm{diag}\left(f'(\boldsymbol{a}^{k-1})\right)\right)\cdot\frac{\partial \boldsymbol{u}^1}{\partial \boldsymbol{w}^1} \tag{8.25}$$

基于同样的推导可得到任意第s层（$1\leqslant s\leqslant K$）的权值向量$\boldsymbol{w}^s$的雅可比矩阵：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^s}=(\boldsymbol{p}-\boldsymbol{y})^{\mathrm{T}}\left(\prod_{k=K}^{s+1}\boldsymbol{W}^k\mathrm{diag}\left(f'(\boldsymbol{a}^{k-1})\right)\right)\cdot\frac{\partial \boldsymbol{u}^s}{\partial \boldsymbol{w}^s} \tag{8.26}$$

其中：

$$\frac{\partial \boldsymbol{u}^s}{\partial \boldsymbol{w}^s}=\begin{pmatrix} x_1^{s-1} & \cdots & x_{n_{s-1}}^{s-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & x_1^{s-1} & \cdots & x_{n_{s-1}}^{s-1} \end{pmatrix}_{n_s\times(n_s\cdot n_{s-1})} \tag{8.27}$$

注意连乘Π是从K开始反着数到$s+1$，若$s=K$则没有连乘项。接下来求交叉熵对各层偏置向量的雅可比矩阵，因为第k层的$\boldsymbol{a}^k=\boldsymbol{u}^k+\boldsymbol{b}^k$，所以：

$$\frac{\partial \boldsymbol{a}^k}{\partial \boldsymbol{b}^k}=\boldsymbol{I} \tag{8.28}$$

于是，交叉熵$\mathcal{L}$对第s层（$1 \leqslant s \leqslant K$）的偏置向量$\boldsymbol{b}^s$的雅可比矩阵是：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^s} = (\boldsymbol{p} - \boldsymbol{y})^{\mathrm{T}} \left(\prod_{k=K}^{s+1} \boldsymbol{W}^k \mathrm{diag}\left(f'(\boldsymbol{a}^{k-1}) \right) \right) \tag{8.29}$$

计算各层的雅可比时不必重头来过，信息积累在连乘项之中，连乘项的反向积累就是反向传播。式（8.26）和式（8.29）就是在多层全连接神经网络计算图上的反向传播公式，它们与第 7 章介绍的特殊反向传播算法是一致的。

8.4　计算图的 Python 实现

本节我们用原生 Python 和 Numpy 库实现计算图以及自动求导，并用计算图搭建多层全连接神经网络。与 TensorFlow 不同，我们的节点不是 2 维、3 维，乃至更高维度的张量，而是向量。根据之前的讨论，原则上只用向量就可以实现任何计算，只是在易用性和效率上打了折扣，但是只以向量为节点能更清晰地展现节点与节点之间的映射关系，以及它们之间的求导关系——雅可比矩阵。首先，我们实现计算图节点的基类，代码如下：

```
import numpy as np

from graph import Graph, default_graph


class Node:
    """
    计算图节点类基类
    """

    def __init__(self, *parents):
        self.parents = parents  # 父节点列表
        self.children = []  # 子节点列表
        self.value = None  # 本节点的值
        self.jacobi = None  # 结果节点对本节点的雅可比矩阵
        self.graph = default_graph  # 计算图对象，默认为全局对象 default_graph

        # 将本节点添加到父节点的子节点列表中
        for parent in self.parents:
            parent.children.append(self)

        # 将本节点添加到计算图中
        self.graph.add_node(self)

    def set_graph(self, graph):
```

```
        """
        设置计算图
        """
        assert isinstance(graph, Graph)
        self.graph = graph

    def get_parents(self):
        """
        获取本节点的父节点
        """
        return self.parents

    def get_children(self):
        """
        获取本节点的子节点
        """
        return self.children

    def forward(self):
        """
        前向传播计算本节点的值，若父节点的值未被计算，则递归调用父节点的 forward 方法
        """
        for node in self.parents:
            if node.value is None:
                node.forward()

        self.compute()

    def compute(self):
        """
        抽象方法，根据父节点的值计算本节点的值
        """
        pass

    def get_jacobi(self, parent):
        """
        抽象方法，计算本节点对某个父节点的雅可比矩阵
        """
        pass

    def backward(self, result):
        """
        反向传播，计算结果节点对本节点的雅可比矩阵
        """
        if self.jacobi is None:
```

```
            if self is result:
                self.jacobi = np.mat(np.eye(self.dimension()))
            else:
                self.jacobi = np.mat(np.zeros((result.dimension(), self.dimension())))

                for child in self.get_children():
                    if child.value is not None:
                        self.jacobi += child.backward(result) * child.get_jacobi(self)

        return self.jacobi

    def clear_jacobi(self):
        """
        清空结果节点对本节点的雅可比矩阵
        """
        self.jacobi = None

    def dimension(self):
        """
        返回本节点的值的向量维数
        """
        return self.value.shape[0] if self.value is not None else None

    def reset_value(self, recursive=True):
        """
        重置本节点的值，并递归重置本节点的下游节点的值
        """

        self.value = None

        if recursive:
            for child in self.children:
                child.reset_value()
```

代码中，Graph 类是计算图类，default_graph 对象是一个全局的计算图对象，它们的实现我们稍后呈现。Node 类是计算图节点的基类，所有类型的节点都继承自 Node 类。Node 类实现了计算图节点的一些公共方法，它的构造函数接受可变数量的 Node 类对象作为本节点的父节点，本节点的值用这些父节点的值计算而得。

节点对象保存父节点和子节点的引用列表，这是构成计算图的“边”的关键数据结构，利用它们，就可以遍历计算图。value 和 jacobi 是节点的属性，分别保存节点的值和某个被视为最终结果的节点对本节点的雅可比矩阵。若它们为空，则表示尚没有被计算。构造函数将通过参数传

进来的节点加入本节点的父节点列表，再将本节点加入所有父节点的子节点列表，最后将本节点加入计算图对象的节点列表。

接下来是一些简单的设置和获取方法，不必赘述。forward 是执行前向传播、计算本节点的值的方法，它是 8.3 节的伪代码的实现。为了计算本节点的值，forward 方法首先检查父节点的值是否为空，若某个父节点的值为空，则递归调用该父节点的 forward 方法。确保所有父节点的值都已被计算后，forward 方法调用 compute 方法计算本节点的值。在基类中，compute 方法是一个抽象方法，需要具体的节点子类去覆盖实现各种不同的计算。get_jacobi 方法是另一个抽象方法，它接受一个父节点，计算当前本节点对这个父节点的雅可比矩阵。

backward 方法是实现反向传播的关键，它接受一个被视为计算图计算结果的节点，计算当前该结果节点对本节点的雅可比矩阵。backward 方法是 8.3 节的伪代码的实现。若本节点的 jacobi 属性为空，则表示结果节点对本节点的雅可比矩阵尚未被计算。若本节点就是结果节点，则雅可比矩阵为单位矩阵，否则利用链式法则根据结果节点对各个子节点的雅可比矩阵计算结果节点对本节点的雅可比矩阵［式（8.17）］。

接下来的两个方法容易理解，不再赘述。reset_value 方法将本节点的值置空。因为本节点的值影响下游节点的值，所以应该递归置空所有下游节点的值。是否递归取决于参数 recursive。

有了基类，我们就可以实现各种不同的节点类，它们执行不同计算。我们首先实现 Variable 类，它保存一个变量。Variable 对象没有父节点，它们是计算图的终端节点。可以随机初始化 Variable 对象的值，也可以为 Variable 对象赋值。Variable 类的代码如下：

```
class Variable(Node):
    """
    变（向）量节点
    """

    def __init__(self, dim, init=False, trainable=True):
        """
        变量节点没有父节点，构造函数接受变量的维数，以及变量是否参与训练的标识
        """
        Node.__init__(self)
        self.dim = dim

        # 如果需要初始化，则以正态分布随机初始化变量的值
        if init:
            self.value = np.mat(np.random.normal(0, 0.001, (self.dim, 1)))
```

```
        # 变量节点是否参与训练
        self.trainable = trainable

    def set_value(self, value):
        """
        为变量赋值
        """
        assert isinstance(value, np.matrix) and value.shape == (self.dim, 1)

        # 本节点的值被改变，重置所有下游节点的值
        self.reset_value()
        self.value = value
```

Variable 类的构造函数接受 dim 参数，确定变量的维数。init 参数表示是否要随机初始化变量的值。trainable 参数表示本变量节点是否参与训练。set_value 方法为 Variable 类独有，它设置变量的值。若变量的值被改变，则计算图中所有下游节点的值都将作废，所以 set_value 方法调用 reset_value 方法递归清除本节点以及所有下游节点的值。Variable 对象的值是被赋予或被随机初始化的，所以它不用实现 compute 方法。Variable 对象没有父节点，它也不用实现 get_jacobi 方法。接下来我们实现向量加法节点，代码如下：

```
class Add(Node):
    """
    向量加法
    """


    def compute(self):
        assert len(self.parents) == 2 and self.parents[0].dimension() == self.parents[1].dimension()
        self.value = self.parents[0].value + self.parents[1].value

    def get_jacobi(self, parent):
        return np.eye(self.dimension())  # 向量之和对其中任一个向量的雅可比矩阵是单位矩阵
```

Add 类的 compute 方法将两个父节点的值相加。get_jacobi 方法求当前 Add 对象对某一个父节点的雅可比矩阵。向量加法是一个$\mathbb{R}^n \to \mathbb{R}^n$的映射，它对其中某一个参与加和的向量的雅可比矩阵是$n \times n$单位矩阵。向量内积（点积）节点的代码如下：

```
class Dot(Node):
    """
    向量内积
    """


    def compute(self):
```

```
        assert len(self.parents) == 2 and self.parents[0].dimension() == self.parents[1].dimension()
        self.value = self.parents[0].value.T * self.parents[1].value  # 1x1 矩阵（标量），为两个父节
点的内积

    def get_jacobi(self, parent):
        if parent is self.parents[0]:
            return self.parents[1].value.T
        else:
            return self.parents[0].value.T
```

在我们的实现中，值一律采用 numpy.matrix 类型，即矩阵。n维向量就是$n \times 1$矩阵，标量就是 1×1 矩阵。Dot 类的 compute 方法计算两个父节点的内积。因为节点的值是 numpy.matrix 类型，经过重载的*运算执行的是矩阵相乘，对于一个行向量（列向量的转置）和一个列向量来说，计算的结果是一个 1×1 矩阵（标量），即这两个向量的内积。get_jacobi 方法计算 Dot 节点对某一个父节点的雅可比矩阵，请看下式：

$$\frac{\partial f(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{j=1}^{n} x_j y_j\right) = y_i \tag{8.30}$$

所以有：

$$\left(\frac{\partial f(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})}{\partial x_1}, \frac{\partial f(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})}{\partial x_2}, \cdots, \frac{\partial f(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})}{\partial x_n}\right) = (y_1, y_2, \cdots, y_n) \tag{8.31}$$

内积对某个向量的雅可比矩阵是另一个向量的转置，这就是 Dot 类的 get_jacobi 方法所返回的值。接下来我们实现 Logistic 节点，它对父节点的每个分量施加 Logistic 函数，代码如下：

```
class Logistic(Node):
    """
    对向量的分量施加 Logistic 函数
    """

    def compute(self):
        x = self.parents[0].value
        self.value = np.mat(1.0 / (1.0 + np.power(np.e, np.where(-x > 1e2, 1e2, -x))))  # 对父节点
的每个分量施加 Logistic

    def get_jacobi(self, parent):
        return np.diag(np.mat(np.multiply(self.value, 1 - self.value)).A1)
```

回忆第 6 章中我们对 Logistic 函数的介绍，可以利用 Logistic 函数的值方便地求得其导数（式 6.11）。Logistic 类的 get_jacobi 方法利用已经计算好的 value 成员计算对父节点的雅可比矩阵。该雅可比矩阵是一个对角矩阵，对角线元素是 Logistic 函数对父节点某个分量的导数。类似

地，ReLU 节点的代码如下：

```
class ReLU(Node):
    """
    对向量的分量施加 ReLU 函数
    """

    def compute(self):
        self.value = np.mat(np.where(self.parents[0].value > 0.0, self.parents[0].value, 0.0))  # 对
父节点的每个分量施加 logistic

    def get_jacobi(self, parent):
        return np.diag(np.where(self.parents[0].value.A1 > 0.0, 1.0, 0.0))
```

ReLU 节点的值和雅可比矩阵都很容易计算，代码自明，不再赘述。我们的计算图只操作向量，没有矩阵乘法，但是通过多个向量内积可以实现矩阵乘法。比如神经网络中常见的仿射计算——用权值矩阵乘以输入向量，这时可以将权值矩阵的每一行作为权值向量，求它们与输入向量的内积，再将多个内积结果组成向量。Vectorize 节点负责将多个父节点的值（都应该是1×1标量）组成一个向量，代码如下：

```
class Vectorize(Node):
    """
    将多个父节点组装成一个向量
    """

    def compute(self):
        assert len(self.parents) > 0
        self.value = np.mat(np.array([node.value for node in self.parents])).T  # 将本节点的父节点
的值列成向量

    def get_jacobi(self, parent):
        return np.mat([node is parent for node in self.parents]).astype(np.float).T
```

对于某一个父节点来说，Vectorize 节点是一个$\mathbb{R}^1 \rightarrow \mathbb{R}^n$的映射，它的雅可比矩阵是一个$n \times 1$矩阵，在对应该父节点位置上的分量为 1，其余分量为 0。接下来，我们实现 SoftMax 节点，代码如下：

```
class SoftMax(Node):
    """
    SoftMax 函数
    """
```

```
    @staticmethod
    def softmax(a):
        a[a > 1e2] = 1e2  # 防止指数过大
        ep = np.power(np.e, a)
        return ep / np.sum(ep)

    def compute(self):
        self.value = SoftMax.softmax(self.parents[0].value)

    def get_jacobi(self, parent):
        """
        我们不实现 SoftMax 节点的 get_jacobi 函数，训练时使用 CrossEntropyWithSoftMax 节点（见下）
        """
        return np.mat(np.eye(self.dimension()))  # 无用
```

SoftMax 节点执行的计算我们已经很熟悉了，但是我们不实现它的 get_jacobi 方法，因为计算 SoftMax 函数对输入向量的雅可比矩阵较复杂，但是如果将 SoftMax 函数的输出送给交叉熵，计算交叉熵损失对 SoftMax 函数的输入向量的雅可比矩阵是相当简单的（式 7.42）。所以我们实现一个将 SoftMax 函数与交叉熵损失合二为一的节点类，代码如下：

```
class CrossEntropyWithSoftMax(Node):
    """
    对第一个父节点施加 SoftMax 之后，再以第二个父节点为标签 One-Hot 向量计算交叉熵
    """

    def compute(self):
        prob = SoftMax.softmax(self.parents[0].value)
        self.value = np.mat(-np.sum(np.multiply(self.parents[1].value, np.log(prob + 1e-10))))

    def get_jacobi(self, parent):
        # 这里存在重复计算，但为了代码清晰简洁，舍弃进一步优化
        prob = SoftMax.softmax(self.parents[0].value)
        if parent is self.parents[0]:
            return (prob - self.parents[1].value).T
        else:
            return (-np.log(prob)).T
```

CrossEntropyWithSoftMax 节点的 compute 方法对第一个父节点的值施加 SoftMax 函数，再与第二个父节点的值计算交叉熵。第二个父节点的值是类别标签的 One-Hot 编码向量。get_jacobi 方法对第一个父节点计算式 7.42，对第二个父节点的雅可比矩阵不会被使用，但是也实现在此。

至此，我们实现了几种典型的计算图节点，它们对于我们接下来要做的事情已经足够。有兴

趣的读者可以自己实现一些其他类型的节点。接下来我们实现 Graph 类，代码如下：

```
class Graph:
    """
    计算图类
    """

    def __init__(self):
        self.nodes = []  # 计算图内的节点的列表

    def add_node(self, node):
        """
        添加节点
        """
        self.nodes.append(node)

    def clear_jacobi(self):
        """
        清除图中全部节点的雅可比矩阵
        """
        for node in self.nodes:
            node.clear_jacobi()

    def reset_value(self):
        """
        重置图中全部节点的值
        """
        for node in self.nodes:
            node.reset_value(False)  # 每个节点不递归清除自己的子节点的值


# 全局默认计算图
default_graph = Graph()
```

我们的 Graph 类较简单，它只保留计算图的全部节点，实现清除所有节点的雅可比矩阵和值的方法。default_graph 是一个全局的 Graph 对象，默认情况下所有节点都将被加入到 default_graph 中。最后，我们实现训练优化器类。所有优化器类都继承自一个基类，代码如下：

```
from graph import Graph, default_graph
from node import *


class Optimizer:
    """
```

```
    优化器基类
    """

    def __init__(self, graph, target, batch_size=12):
        assert isinstance(target, Node) and isinstance(graph, Graph)
        self.graph = graph
        self.target = target
        self.batch_size = batch_size

        # 为每个参与训练的节点累加一个 Mini Batch 的全部样本的梯度
        self.acc_gradient = dict()
        self.acc_no = 0

    def one_step(self):
        """
        计算并累加样本的梯度，一个 Mini Batch 结束后执行变量更新
        """
        self.forward_backward()

        self.acc_no += 1
        if self.acc_no >= self.batch_size:
            self.update()
            self.acc_gradient.clear()  # 清除梯度累加
            self.acc_no = 0  # 清除计数

    def get_gradient(self, node):
        """
        返回一个 Mini Batch 的样本的平均梯度
        """
        assert node in self.acc_gradient
        return self.acc_gradient[node] / self.batch_size

    def update(self):
        """
        抽象方法，利用梯度更新可训练变量
        """

        pass

    def forward_backward(self):
        """
        前向传播计算结果节点的值并反向传播计算结果节点对各个节点的梯度
        """

        # 前向传播计算结果节点
```

```
        self.target.forward()

        # 反向传播计算梯度
        for node in self.graph.nodes:
            if isinstance(node, Variable) and node.trainable:
                node.backward(self.target)

                if node not in self.acc_gradient:
                    self.acc_gradient[node] = node.jacobi.T
                else:
                    self.acc_gradient[node] += node.jacobi.T

        # 更新完毕，清除计算图中所有节点的雅可比矩阵
        self.graph.clear_jacobi()
```

Optimizer 类的构造函数接受一个 Graph 对象、一个作为优化目标的节点对象，以及 Mini Batch 的样本数量。因为我们的计算图节点只能包含一个向量，所以不能利用更高的维度在节点值中包含整个 Mini Batch。于是，我们对 Mini Batch 的实现是这样的：对一个 Mini Batch 中的样本依次执行前向传播和反向传播，将参与训练的变量的梯度累加在 acc_gradient 中，一个 Mini Batch 计算完毕后执行变量更新，这时使用 Mini Batch 中多个样本的平均梯度。

one_step 方法调用 forward_backward 方法对一个样本执行前向传播和反向传播，将目标节点对各个变量的梯度累加在 acc_gradient 中，最后清除所有节点的雅可比矩阵。one_step 方法计数样本数，当样本数达到 batch_size 时，执行 update 方法更新变量，并清除累加梯度以及计数。get_gradient 方法返回一个 Mini Batch 的所有样本的平均梯度。

update 是抽象方法，利用 Mini Batch 的平均梯度以各种不同的方法更新变量值。update 方法将在具体的优化器类中得到实现。forward_backward 方法执行前向传播，计算目标节点的值，然后反向传播计算目标节点对每个参与训练的变量节点的雅可比矩阵。原始梯度下降的优化器实现如下：

```
class GradientDescent(Optimizer):
    """
    梯度下降优化器
    """

    def __init__(self, graph, target, learning_rate=0.01, batch_size=32):
        Optimizer.__init__(self, graph, target, batch_size)
        self.learning_rate = learning_rate

    def update(self):
```

```
        for node in self.graph.nodes:
            if isinstance(node, Variable) and node.trainable:
                gradient = self.get_gradient(node)

                node.set_value(node.value - self.learning_rate * gradient)
```

除了 Optimizer 类的参数，GradientDescent 类的构造函数还接受 learning_rate 参数，即学习率。GradientDescent 类的 update 方法相当简单，就是获得平均梯度，乘以学习率并取反后更新到变量节点的当前值上。RMSProp 和 Adam 优化器的代码如下：

```
class RMSProp(Optimizer):
    """
    RMSProp 优化器
    """

    def __init__(self, graph, target, learning_rate=0.01, beta=0.9, batch_size=32):
        Optimizer.__init__(self, graph, target, batch_size)
        self.learning_rate = learning_rate

        assert 0.0 < beta < 1.0
        self.beta = beta

        self.s = dict()

    def update(self):

        for node in self.graph.nodes:
            if isinstance(node, Variable) and node.trainable:
                gradient = self.get_gradient(node)

                if node not in self.s:
                    self.s[node] = np.power(gradient, 2)
                else:
                    self.s[node] = self.beta * self.s[node] + (1 - self.beta) * np.power(gradient, 2)

                node.set_value(node.value - self.learning_rate * gradient / (np.sqrt(self.s[node]
                    + 1e-10)))


class Adam(Optimizer):
    """
    Adam 优化器
    """
```

```
    def __init__(self, graph, target, learning_rate=0.01, beta_1=0.9, beta_2=0.99, batch_size=32):
        Optimizer.__init__(self, graph, target, batch_size)
        self.learning_rate = learning_rate

        assert 0.0 < beta_1 < 1.0
        self.beta_1 = beta_1

        assert 0.0 < beta_2 < 1.0
        self.beta_2 = beta_2

        self.s = dict()
        self.v = dict()

    def update(self):

        for node in self.graph.nodes:
            if isinstance(node, Variable) and node.trainable:
                gradient = self.get_gradient(node)

                if node not in self.s:
                    self.v[node] = gradient
                    self.s[node] = np.power(gradient, 2)
                else:
                    self.v[node] = self.beta_1 * self.v[node] + (1 - self.beta_1) * gradient
                    self.s[node] = self.beta_2 * self.s[node] + (1 - self.beta_2) * np.power
                        (gradient, 2)

                node.set_value(node.value - self.learning_rate * self.v[node] / np.sqrt
                    (self.s[node] + 1e-10))
```

8

关于 RMSProp 和 Adam，前文已经有了详细论述和 Python 实现，这里只是把相同逻辑实现在我们的计算图优化器框架内，就不再详细解释了。有兴趣的读者可以自己实现其他优化器。最后，我们用这个计算图框架搭建一个多层全连接神经网络，并用它分类鸟类生态类群，代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from node import *
from optimizer import *
```

```
bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)

# 先将 6 个类别的字符串名称编码成整数编号，再转化成 6 个 1/0 值的 one hot 编码
label_encoder = LabelEncoder()
one_hot_label = OneHotEncoder(sparse=False, categories="auto").fit_transform(
    label_encoder.fit_transform(bird.type).reshape(-1, 1))

# 去掉类别的字符串名称列
bird.drop("type", axis=1, inplace=True)

# 将 10 个数值型特征列与 6 个类别标签 one hot 编码列合并起来
data = np.c_[bird.values, one_hot_label]

# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集，将特征与 one hot 编码分开
ss = StandardScaler()
train_x = ss.fit_transform(np.mat(data[:300, :-6]))
train_y = np.mat(data[:300, -6:])

# 其余样本作为测试集，将特征与 one hot 编码分开
test_x = ss.transform(np.mat(data[300:, :-6]))
test_y = np.mat(data[300:, -6:])

# 构造多层全连接神经网络的计算图
X = Variable(10, trainable=False)  # 10 维特征变量

# 隐藏层 12 个神经元
hidden_layer = []
for i in range(12):
    hidden_layer.append(Add(Dot(Variable(10, True), X), Variable(1, True)))

# 隐藏层的输出
hidden_layer_output = ReLU(Vectorize(*hidden_layer))

# 输出层 6 个神经元
output_layer = []
for i in range(6):
    output_layer.append(Add(Dot(Variable(12, True), hidden_layer_output), Variable(1, True)))

# 输出层的仿射值以及施加 SoftMax 后的概率值
output = Vectorize(*output_layer)
prob = SoftMax(output)
```

```
# 训练标签
label = Variable(6, trainable=False)

# 交叉熵损失
loss = CrossEntropyWithSoftMax(output, label)  # 注意第一个父节点是输出层的仿射值

# Adam 优化器
optimizer = Adam(default_graph, loss, 0.05, batch_size=32)

# 训练
for e in range(500):

    # 每个 epoch 开始时在测试集上评估模型正确率
    probs = []
    losses = []
    for i in range(len(test_x)):
        X.set_value(np.mat(test_x[i, :]).T)
        label.set_value(np.mat(test_y[i, :]).T)

        # 前向传播计算概率
        prob.forward()
        probs.append(prob.value.A1)

        # 计算损失值
        loss.forward()
        losses.append(loss.value[0, 0])

    # 取概率最大的类别为预测类别
    pred = [label_encoder.classes_[idx] for idx in np.argmax(np.array(probs), axis=1)]
    truth = [label_encoder.classes_[idx] for idx in np.argmax(test_y, axis=1).A1]
    accuracy = accuracy_score(truth, pred)

    print("Epoch：{:d}，损失值：{:.3f}，正确率：{:.2f}%".format(e + 1, np.mean(losses), accuracy * 100))

    for i in range(len(train_x)):
        X.set_value(np.mat(train_x[i, :]).T)
        label.set_value(np.mat(train_y[i, :]).T)

        # 执行一步优化
        optimizer.one_step()

# 训练结束后打印最终评价
print("测试集正确率：{:.3f}".format(accuracy_score(truth, pred)))
print(classification_report(truth, pred))
```

```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111)
_ = sns.heatmap(
    confusion_matrix(truth, pred),
    square=True,
    xticklabels=label_encoder.classes_,
    annot=True,
    annot_kws={"fontsize": 8},
    yticklabels=label_encoder.classes_,
    cbar=False,
    cmap=sns.light_palette("#00304e", as_cmap=True),
    ax=ax
)
```

准备数据以及模型评价的过程与之前的代码一样。构造多层全连接神经网络计算图的关键代码如下：

```
# 构造多层全连接神经网络的计算图
X = Variable(10, trainable=False)  # 10 维特征变量

# 隐藏层 12 个神经元
hidden_layer = []
for i in range(12):
    hidden_layer.append(Add(Dot(Variable(10, True), X), Variable(1, True)))

# 隐藏层的输出
hidden_layer_output = ReLU(Vectorize(*hidden_layer))

# 输出层 6 个神经元
output_layer = []
for i in range(6):
    output_layer.append(Add(Dot(Variable(12, True), hidden_layer_output), Variable(1, True)))

# 输出层的仿射值以及施加 SoftMax 后的概率值
output = Vectorize(*output_layer)
prob = SoftMax(output)

# 训练标签
label = Variable(6, trainable=False)

# 交叉熵损失
loss = CrossEntropyWithSoftMax(output, label)  # 注意第一个父节点是输出层的仿射值
```

X 是一个 10 维变（向）量，对应样本的 10 个特征。hidden_layer 数组保存隐藏层的 12 个神

经元。它们用 Add、Dot 与 Variable 节点构造，其中 Variable 类型节点保存 12 个 10 维权值向量和偏置。接下来用 Vectorize 节点将 hidden_layer 数组中的节点组成向量，施加 ReLU 激活函数，这就得到了隐藏层的输出节点 hidden_layer_output。

output_layer 数组保存输出层的 6 个神经元，仍然用 Add、Dot 与 Variable 节点构造，但权值向量是 12 维的。将 output_layer 数组中的节点组成向量后得到 output 节点，以它为父节点构造 SoftMax 节点，即为神经网络的输出节点。但为了训练，我们以 output 节点为父节点构造 CrossEntropyWithSoftMax 节点 loss。除了 output 节点，loss 的父节点还包括 label 节点，它是一个 6 维变（向）量节点，训练时被赋予样本类别标签 One-Hot 编码。该神经网络计算图的训练代码如下：

```
# Adam 优化器
optimizer = Adam(default_graph, loss, 0.05, batch_size=32)

# 训练
for e in range(500):
    for i in range(len(train_x)):
        X.set_value(np.mat(train_x[i, :]).T)
        label.set_value(np.mat(train_y[i, :]).T)

        # 执行一步优化
        optimizer.one_step()
```

首先构造一个 Adam 优化器对象 optimizer，它接受 loss 节点为优化目标，学习率取 0.05，一个 Mini Batch 包含 32 个样本，β^1和β^2参数都取默认值。训练进行 500 个 epoch，每个 epoch 依次将训练样本赋给变量节点 X，执行 optimizer 的 one_step 函数，更新多层全连接神经网络的各个权值向量和偏置。

为了展现计算图的表达能力，我们再搭建两个非全连接的神经网络。首先，我们搭建推荐系统和 CTR 预估领域常用的 Wide & Deep 模型，代码如下：

```
# 输入向量：一次特征
X = Variable(10, trainable=False)  # 10 维特征变量

# 输入向量：二次特征，共 C(10,2)=45 个
X_2 = Variable(45, trainable=False)

# Wide 部分
wide = []
for i in range(6):
```

```
    wide.append(Add(Dot(Variable(45, True), X_2), Variable(1, True)))
wide = Vectorize(*wide)

# Deep 部分
hidden1_size = 12
hidden2_size = 12

# 第 1 隐藏层的神经元
hidden1 = []
for i in range(hidden1_size):
    hidden1.append(Add(Dot(Variable(10, True), X), Variable(1, True)))

# 第 1 隐藏层的输出
hidden1 = ReLU(Vectorize(*hidden1))

# 第 2 隐藏层的神经元
hidden2 = []
for i in range(hidden2_size):
    hidden2.append(Add(Dot(Variable(hidden1_size, True), hidden1), Variable(1, True)))

# 第 2 隐藏层的输出
hidden2 = ReLU(Vectorize(*hidden2))

# 输出层的 6 个神经元
deep = []
for i in range(6):
    deep.append(Add(Dot(Variable(hidden2_size, True), hidden2), Variable(1, True)))
deep = Vectorize(*deep)

# 输出层的仿射值以及施加 SoftMax 后的概率值
logits = Add(wide, deep)
prob = SoftMax(logits)

# 训练标签
label = Variable(6, trainable=False)

# 交叉熵损失
loss = CrossEntropyWithSoftMax(logits, label)  # 注意第一个父节点是输出层的仿射值
```

X_2 是 45 维变（向）量，因为 10 个特征能产生$C_{10}^2 = 45$个二次特征，例如x_7x_9。从数学角度看，引入二次项作为新特征，可以将线性模型提升为二次非线性模型；从业务角度看，新引入的二次项描绘了特征与特征之间的交互作用。X_2 容纳新引入的二次项。

用 45 个二次项搭建一个简单的 6 分类逻辑回归，它的输出（未施加 SoftMax）是向量 wide。

接着，我们构造了一个双隐藏层的全连接神经网络。该网络的输入是 10 维变（向）量 X，接受样本的 10 个一次特征。两个隐藏层各有 12 个神经元，激活函数为 ReLU。输出层包含 6 个神经元，对应 6 个类别。它们的输出（未施加 SoftMax）是 deep 向量。将 wide 和 deep 相加后施加 SoftMax，就得到了 Wide & Deep 模型的输出。将 wide 和 deep 相加后送给 CrossEntropyWithSoftMax 节点，就得到了训练时的交叉熵损失。

注意，我们实现的 Wide & Deep 模型与原始论文中的模型有细节上的出入，但是它把握住了核心思想，体现了 Wide & Deep 模型作为非全连接神经网络的本质，并展现了计算图的表达能力。将 Wide & Deep 模型应用于鸟类生态类群问题的训练和评估代码详见本书网络资源，这里不再赘述。

最后，我们用因子分解机（factorization machine，FM）的思想改造上面的 Wide & Deep 模型，将 Wide 部分替换为一个因子分解机，代码如下：

```
# fm 部分，隐藏向量
fm = []
for s in range(6):
    hidden_vectors = []
    for i in range(10):
        hidden_vectors.append(Variable(6, True))

    # factors
    factors = []
    for i in range(10):
        for j in range(i):
            factors.append(Dot(hidden_vectors[i], hidden_vectors[j]))
    factors = Vectorize(*factors)

    fm.append(Add(Dot(factors, X_2), Variable(1, True)))

fm = Vectorize(*fm)
```

用上述代码替换 Wide & Deep 模型的 Wide 部分，就将二次项的逻辑回归改造成了因子分解机。因子分解机为每一个特征准备一个隐藏向量，针对每一个二次项$x_i x_j$，将x_i的隐藏向量h_i和x_j的隐藏向量h_j的内积$h_i^{\mathrm{T}} h_j$作为$x_i x_j$项的权重，构造二次项的逻辑回归模型。这样做的好处我们不再详细阐述，读者可以查阅因子分解机相关的介绍。

将 Wide & Deep 模型的 Wide 部分替换为因子分解机后所形成的模型，我们可以叫它 FM & Deep 模型。这是我们新提出的模型，学术界和产业界之前没有这种模型。这个例子展现了计算

图的能力，读者可以运用计算图搭建并训练任意结构的新模型，比如 FM & Deep。FM & Deep 其实是沿着 Wide & Deep 的深浅结合的思路，尝试用 FM 增强 Wide 部分。这就是 FM & Deep 模型的动机，至于这个模型是否有好的效果，则需要实验的验证。实际上，学术界和产业界已经有沿着这条思路设计并经过验证的模型了，比如 DeepFM，读者可以自行查阅相关论文。

本节我们实现了一个简单的计算图框架，其功能并不完善，效率也远非最优，主要目的是使读者更清晰透彻地理解计算图的原理，深切体会计算图搭建和训练任意网络结构的能力。第 7 章介绍的反向传播是一种特设（ad hoc）的算法，它只适用于多层全连接神经网络，而有了计算图工具，就可以随心所欲地尝试各种网络结构、损失函数以及正则项，为建模试验打开广阔空间的大门。

8.5　小结

本章介绍了计算图，大部分神经网络都可以用计算图表示。以计算图中的一个节点为最终结果，可以计算它对其他节点的雅可比，这就是计算图的自动求导。在神经网络语境下，自动求导可看作是广义的反向传播。

经过前面若干章节的逐步铺垫，我们于本章达到高峰。我们看到，大部分人工神经网络都可以用计算图描述，神经网络的训练就是计算图上的自动求导再加梯度下降。本书后续章节将进入深度学习领域，在那里，我们将遇到各种复杂的神经元连接方式，以及各种大规模的深度神经网络，只要掌握了本章介绍的工具，就掌握了构建和训练任意神经网络的方法，从此踏上胜利的坦途。

第 9 章

卷积神经网络



上一章我们遇到了一种非全连接神经网络，它有两层神经元：隐藏层和输出层。隐藏层与输出层之间是全连接的，但是隐藏层神经元并不连接全部输入。如果将输入看成二维网格，则每个隐藏层神经元连接到一个子网格，全体子网格覆盖全部输入。这种连接方式其实就是卷积层，这个神经网络就是一个卷积神经网络（convolutional neural network，CNN）。本章我们正式介绍卷积神经网络。

我们首先介绍卷积及其在图像处理领域的应用，并从“可训练的滤波器”角度引入卷积层。之后，我们介绍 CNN 的主要组件：卷积层、激活层、池化层和全连接层。CNN 由这些层叠加而成，叠加的方式多种多样，形成各种不同的 CNN。CNN 可能具有几十乃至上百层，从而跨入深度学习领域。深度学习会遇到诸如梯度消失、内部协变量漂移、过拟合等问题，人们发明了一些技术应对这些问题，包括权值初始化、Dropout、数据增强、批标准化以及残差学习等。本章的最后，我们讨论这些深度学习特有的问题和技术。

9.1　卷积

CNN 因卷积而得名，故本节首先介绍卷积。卷积是一种针对函数的运算，它用两个函数得到一个新函数。本节介绍卷积运算的原理和含义，并着重介绍其在图像处理中的应用，这也是卷积层作为 CNN 最重要的组件而发挥作用的原因。

9.1.1　一元函数的卷积

两个一元函数$f\colon \mathbb{R}\to\mathbb{R}$和$g\colon \mathbb{R}\to\mathbb{R}$的卷积（convolution）是一个新的函数：

$$(f * g)(x)=\int_{-\infty}^{\infty} f(t)g(x-t)\mathrm{d}t \tag{9.1}$$

卷积$f*g$是关于x的函数，它将$f(t)g(x-t)$对t在实数轴上积分。把这个积分写成黎曼和的极限的形式，可以更清楚地洞察卷积的意义：

$$(f*g)(x)=\lim_{\max(\Delta t_i)\to 0}\sum_{i=-\infty}^{\infty}f(t_i)g(x-t_i)\Delta t_i \tag{9.2}$$

将数轴分成无数小区间，各个小区间的宽度是Δt_i，在每个小区间内取一个值t_i，对所有小区间求$f(t_i)g(x-t_i)\Delta t_i$之和，即黎曼和。式（9.2）的含义是：卷积$f*g$在x的值，是函数g在$(x-t_i)$的值以$f(t_i)\Delta t_i$为权重的加权和在区间宽度Δt_i趋近于 0 时的极限，如图 9-1 所示。

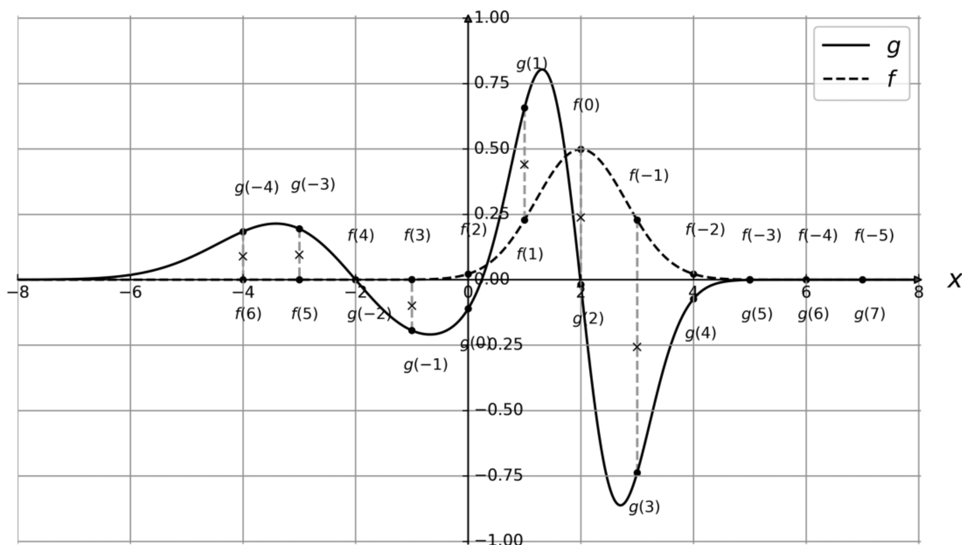


图 9-1　$f*g$在x（$=2$）的值是加权和在采样间隔趋近于零时的极限

如果f是一个钟形函数，例如标准正态分布的概率密度函数：

$$f(t)=\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{t^2}{2}} \tag{9.3}$$

则f和g的卷积是：

$$(f*g)(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\mathrm{e}^{-\frac{t^2}{2}}\cdot g(x-t)\mathrm{d}t \tag{9.4}$$

$f*g$是函数g的所有值的加权积分。f在原点最大，沿着正负两侧衰减并趋近于 0。对于x，f赋予x最大权重，距离x越远则权重越小。$f*g$的效果是对函数g做模糊（blur）——称为高斯模糊。若g是随机函数，则$f*g$的图像如图 9-2 所示。

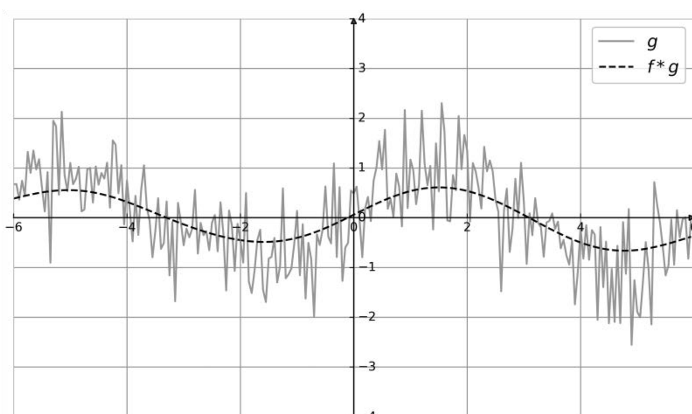


图 9-2 高斯模糊的效果

以 0 均值、σ标准差的正态分布概率密度函数作为f：

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{t^2}{2\sigma^2}} \tag{9.5}$$

则可以通过标准差控制模糊的程度：标准差越大，权重分布越分散，模糊效果越强；标准差越小，权重分布越集中，模糊效果越弱。标准差趋近于 0 时，$f(t)$趋近于狄拉克函数（Dirac function）——只在原点不为 0，且在实数轴上积分为 1。这种情况下$(f * g)(x) = g(x)$，没有模糊效果。不同标准差的高斯模糊的效果如图 9-3 所示。

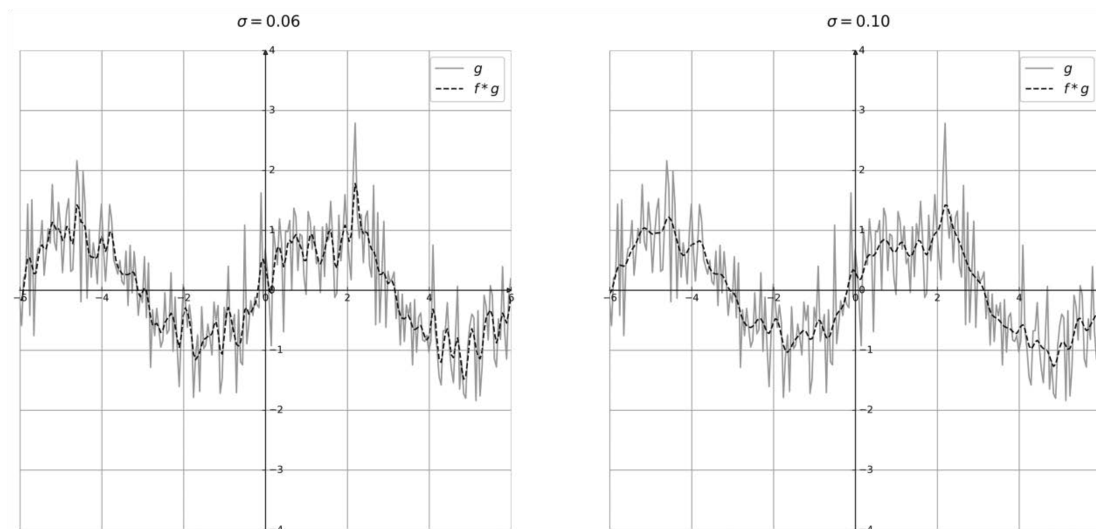


图 9-3 不同标准差的高斯模糊的效果

9

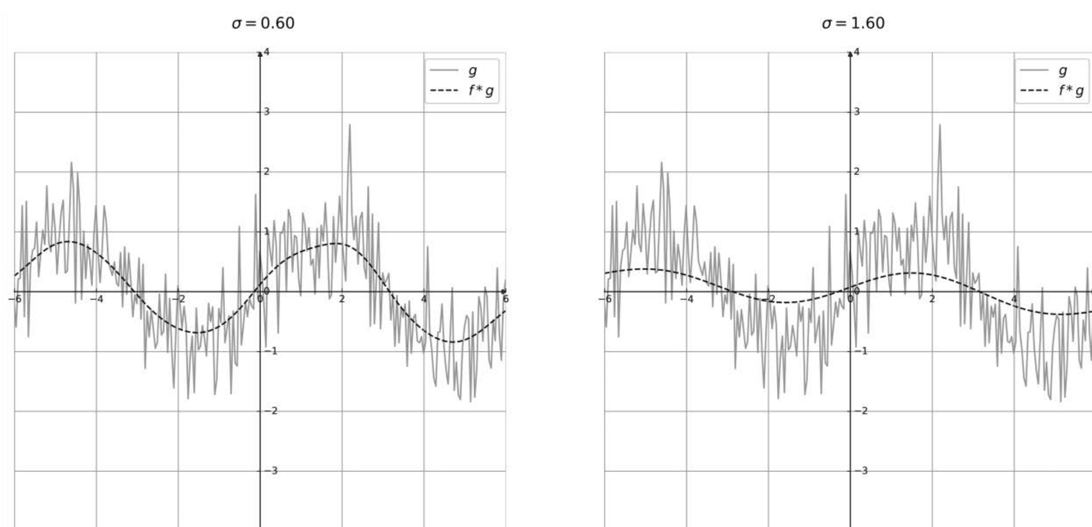


图 9-3　（续）

如果式(9.2)中黎曼和的小区间的边界点都是整数，并且t_i取小区间的边界，即$0, \pm 1, \pm 2, \cdots$，则所有Δt都为 1，黎曼和是：

$$(f * g)(\boldsymbol{x}) \approx \sum_{t=-\infty}^{\infty} f(t) g(x-t) \tag{9.6}$$

式（9.6）是离散卷积，f和g可以是离散函数，只要求它们在整数值上有定义，如图 9-4 所示。

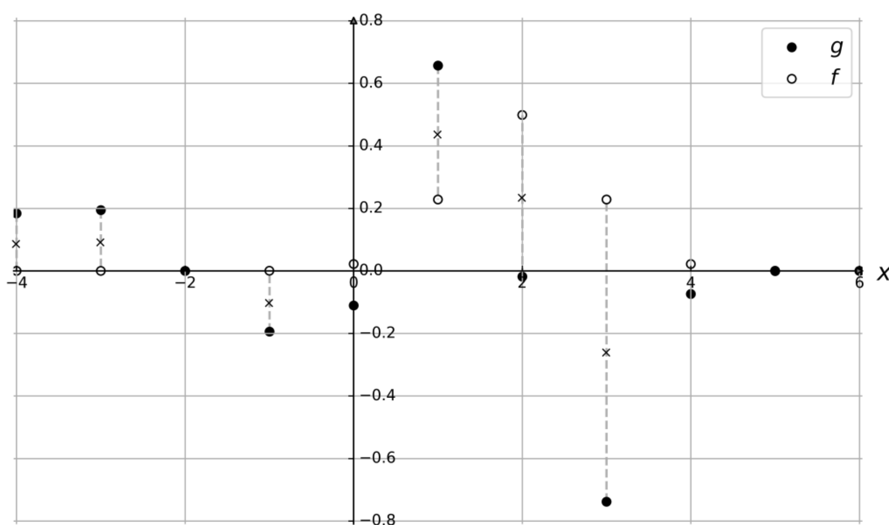


图 9-4　f和g的离散卷积在x（$=2$）的值是加权和

9.1.2　多元函数的卷积

两个n元函数$f:\mathbb{R}^n\to\mathbb{R}$和$g:\mathbb{R}^n\to\mathbb{R}$的卷积是一个新的$n$元函数：

$$(f*g)(\boldsymbol{x})=\int_{\mathbb{R}^n}f(\boldsymbol{t})g(\boldsymbol{x}-\boldsymbol{t})\mathrm{d}\boldsymbol{t}=\int\int\cdots\int f(\boldsymbol{t})g(\boldsymbol{x}-\boldsymbol{t})\mathrm{d}t_1\mathrm{d}t_2\cdots\mathrm{d}t_n \tag{9.7}$$

这是式（9.1）的多元形式：遍历全部$\boldsymbol{t}\in\mathbb{R}^n$，以微元$f(\boldsymbol{t})\mathrm{d}\boldsymbol{t}$为权值，对$g$在$\boldsymbol{x}-\boldsymbol{t}$的值加权求“和”（积分意义上）。令$f$是二元正态分布$\mathcal{N}(\boldsymbol{0},\sigma^2\boldsymbol{I})$的概率密度函数：

$$f(\boldsymbol{t})=\frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{\|\boldsymbol{t}\|^2}{2\sigma^2}} \tag{9.8}$$

f在原点最大，随着$\boldsymbol{t}$远离原点而衰减。σ^2越大则衰减越慢，权重分布越分散；σ^2越小则衰减越快，权重分布越集中。$f*g$的值是函数g的所有值的加权“和”，所以$f*g$是对g的二维高斯模糊。二元积分是二元黎曼和的极限，将二维平面切分成1×1网格，(t_1,t_2)取网格的角，这种情况下二元黎曼和是：

$$(f*g)(\boldsymbol{x})\approx\sum_{t_2=-\infty}^{\infty}\sum_{t_1=-\infty}^{\infty}f(t_1,t_2)g(x_1-t_1,x_2-t_2) \tag{9.9}$$

式（9.9）是二元离散卷积，它要求f和g在二维整数网格上有定义。令f为图 9-5 所示的二元离散函数，图中未显示的位置上的值为 0。

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

图 9-5　二元离散函数

这个二元离散函数在中心（原点）的值最大，向四周逐渐衰减，在以原点为中心的 5×5 区域外函数值衰减为 0。它其实是标准二元正态分布$\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$在整数值上的采样，将 5×5 区域外的值忽略。$f$称为二元离散高斯函数。令$g$为图 9-6 所示的二元离散函数。

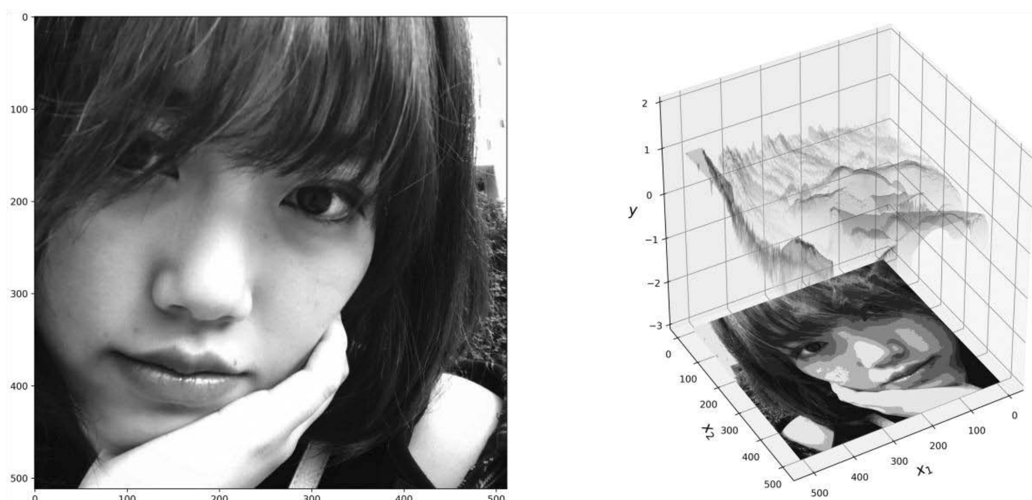


图 9-6　作为图像的二元离散函数

二元离散函数g是一幅图像。g的值在$[0,1]$区间内，1 为白色，0 为黑色，0 与 1 之间的值为不同灰度的灰色。g在 512×512 区域以外的值为 0。根据式（9.9），f和g的离散卷积在$\boldsymbol{x}$的值，是将f的中心（原点）对准$\boldsymbol{x}$，将镜像翻转的f与函数g在对应位置上的值相乘并加和，如图 9-7 所示。

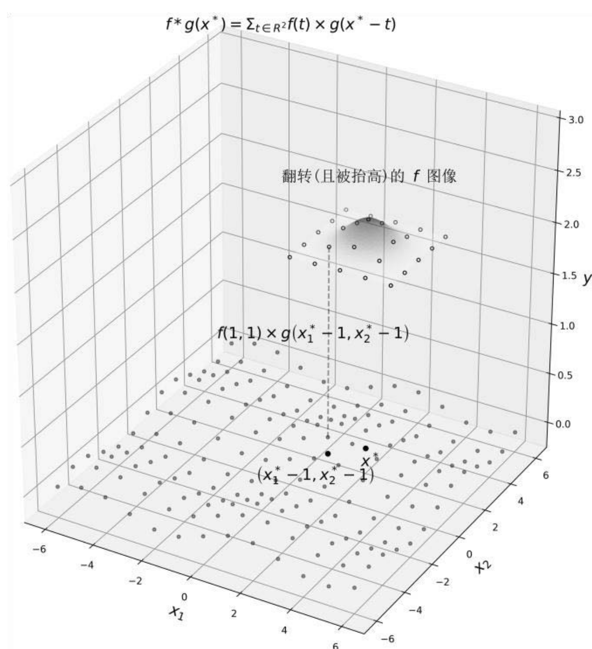


图 9-7　二元离散卷积

为了论述简洁，后文省去镜像翻转。只要记住，后文中的f其实是真正参与卷积运算的f的镜像翻转就可以了。图 9-5 中的二元离散高斯函数（不翻转）与图 9-6 中的g的卷积如图 9-8 所示。

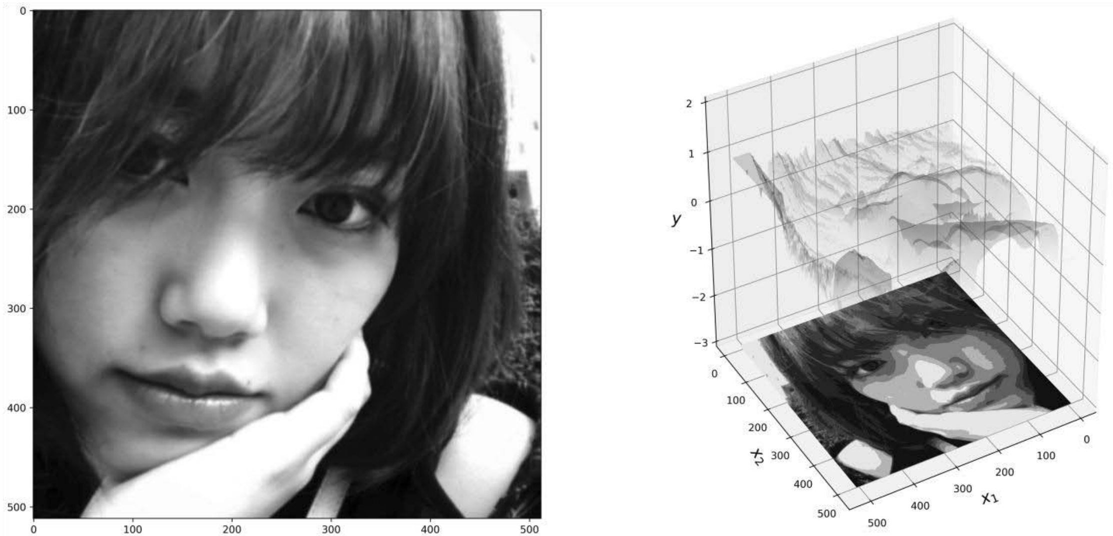


图 9-8 卷积图像

可以看到，这个卷积运算对图像产生模糊效果。我们还可以使用不同方差的二元离散高斯函数，如图 9-9 所示。

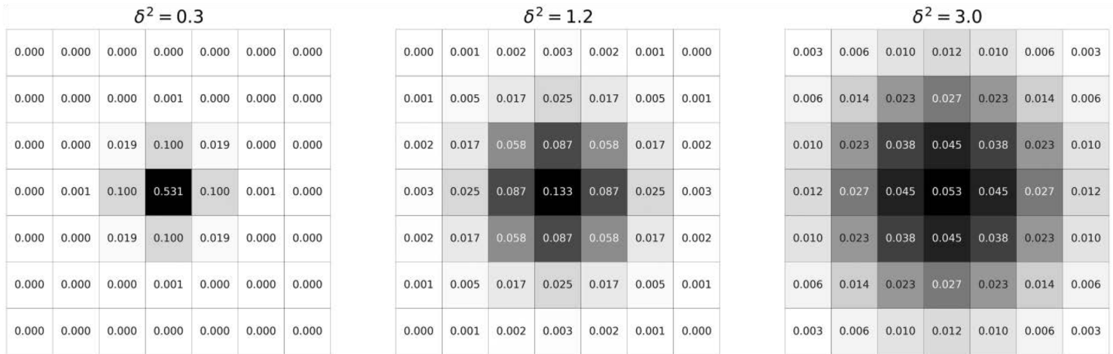


图 9-9 不同方差的二元离散高斯函数

方差越小，权重分布越集中，模糊效果越弱；方差越大，权重分布越分散，模糊效果越强。图 9-9 中的三个二元离散高斯函数与函数g的卷积如图 9-10 所示。

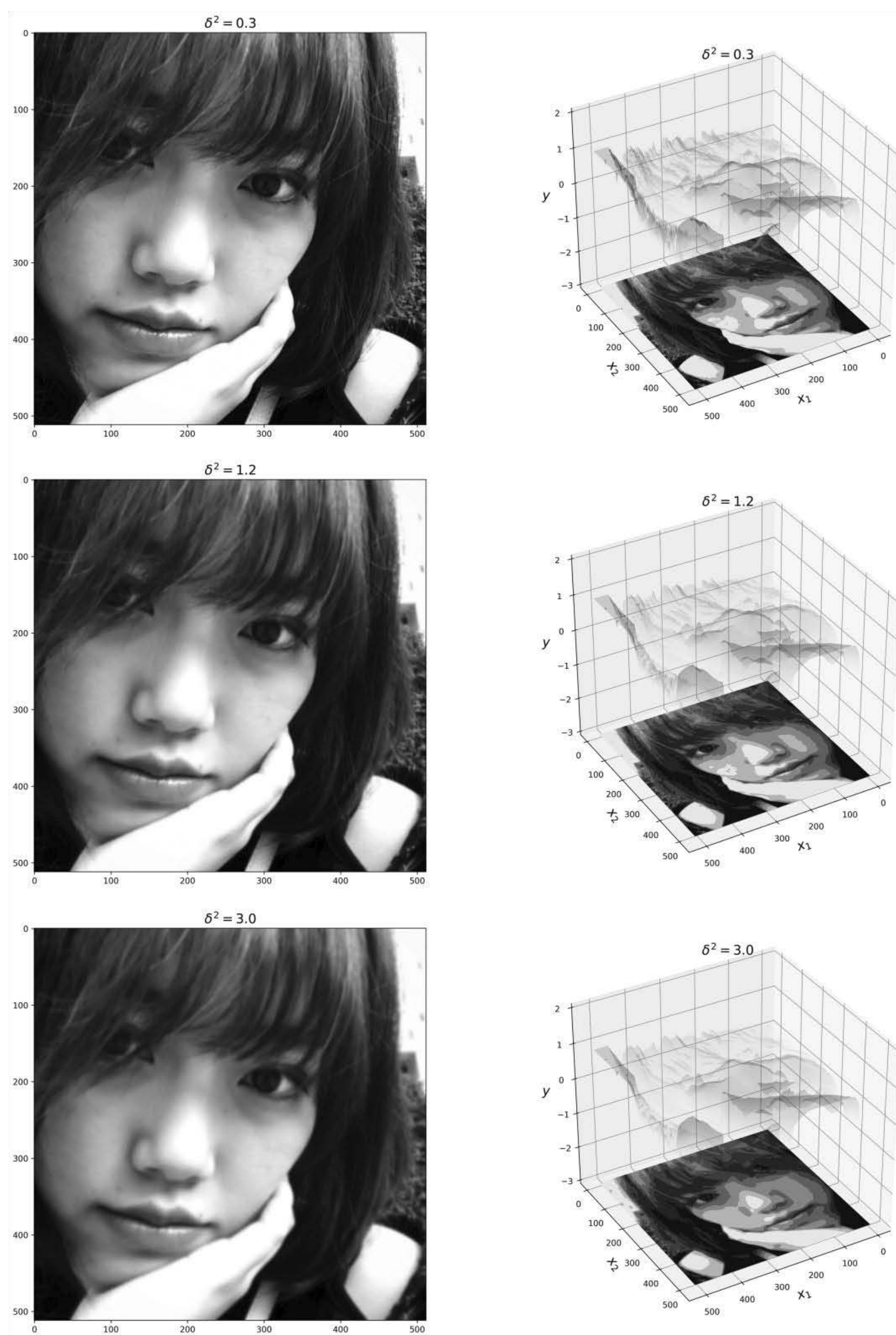


图 9-10　不同方差的二元离散高斯函数与图像的卷积

9.1.3　滤波器

在数字图像处理领域，离散卷积中的f称作卷积核（kernel）或滤波器（filter）。f有值的区域的大小称作卷积核或滤波器的尺寸（size），例如 3×3、5×5 等。离散卷积操作称为滤波（filtering）。我们已经认识了高斯滤波器（Gaussian filter），它起到模糊的作用。PhotoShop 就有高斯模糊功能。除高斯滤波器外，还有其他很多滤波器，它们各具不同的功能。本节介绍几种常见的滤波器。

在连续的情况下，拉普拉斯算子是函数在x和y两个方向的二阶偏导数之和。图 9-11 中的滤波器是二元离散拉普拉斯滤波器，它增强函数（图像）的二阶导数较大的区域，对于图像中的细线条具有强化效果。

-1	-1	-1
-1	8	-1
-1	-1	-1

图 9-11　3×3 离散拉普拉斯滤波器

用图 9-11 中的滤波器对函数g做滤波，效果如图 9-12 所示。

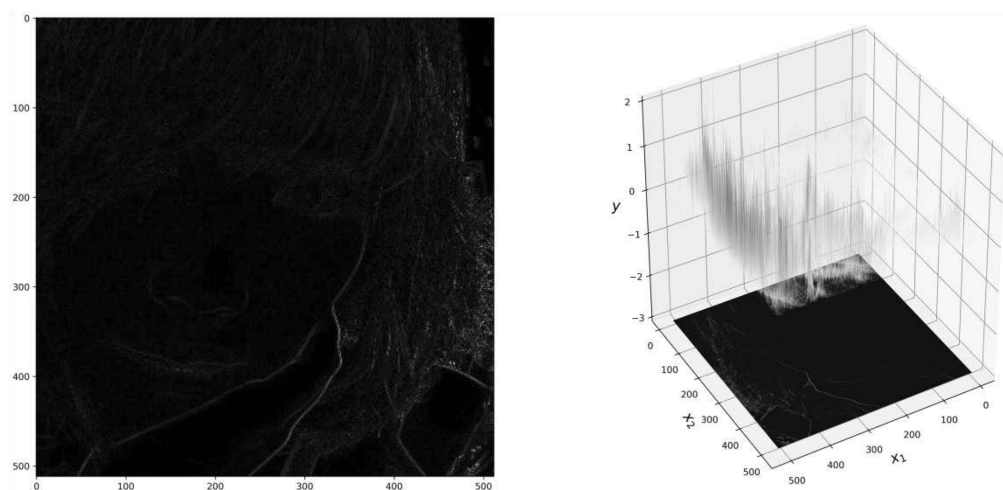


图 9-12　3×3 离散拉普拉斯滤波器的效果

方框模糊（box blur）滤波器如图 9-13 所示，它通过将周围像素取平均而达到模糊效果。

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

图 9-13　5 × 5 方框模糊滤波器

用图 9-13 中的方框模糊滤波器对函数g做滤波，效果如图 9-14 所示。

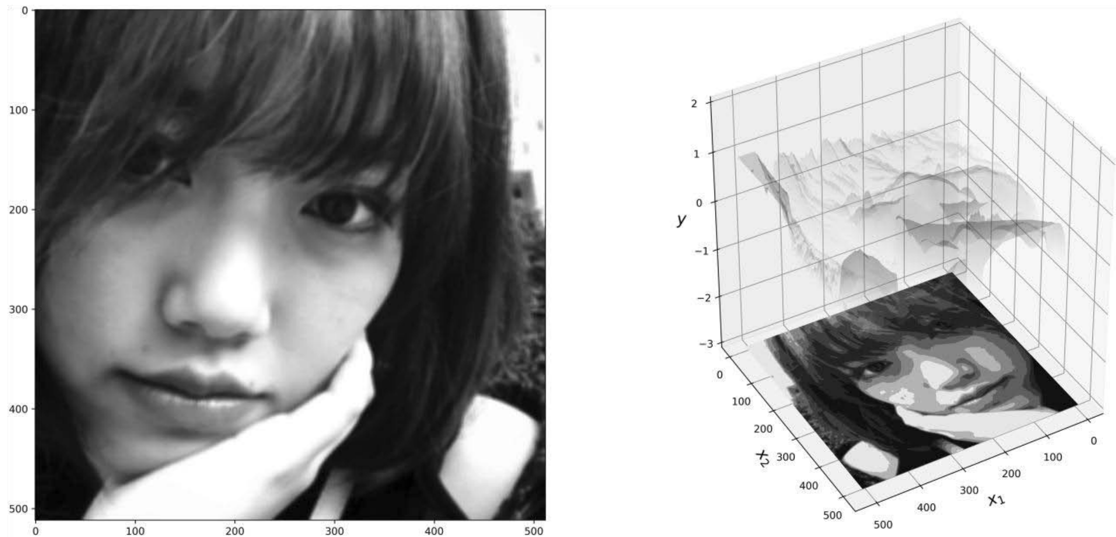


图 9-14　5 × 5 方框模糊的效果

Sobel 滤波器的功能是检测图像中的边缘。Sobel 滤波器分为纵向和横向两种，分别用来检测图像中的纵向和横向边缘，如图 9-15 所示。

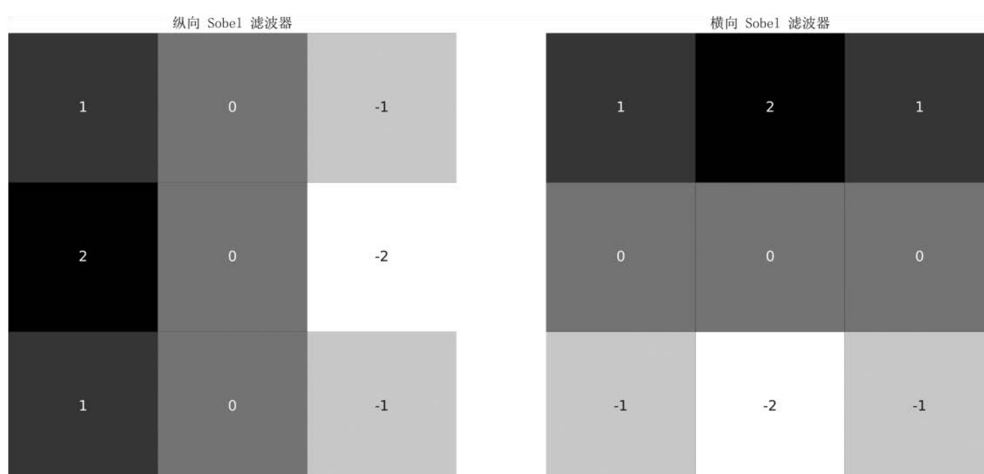


图 9-15 纵向和横向 Sobel 滤波器

Sobel 滤波器是离散差分算子。以纵向 Sobel 滤波器为例，它对当前位置左侧的值施加正系数，对右侧的值施加负系数。若像素位于灰度值横向变化平缓的区域，它左右两侧的灰度值相差较小，这种情况下，纵向 Sobel 滤波器将两侧抵消，得到接近 0 的结果。若像素位于两个区域之间的纵向边缘上，它左右两侧的灰度值相差较大，这种情况下，纵向 Sobel 滤波器会得到绝对值较大的结果。纵向 Sobel 滤波器得到函数横向的方向导数的近似。对函数g施加纵向和横向 Sobel 滤波器后取绝对值，效果如图 9-16 所示。

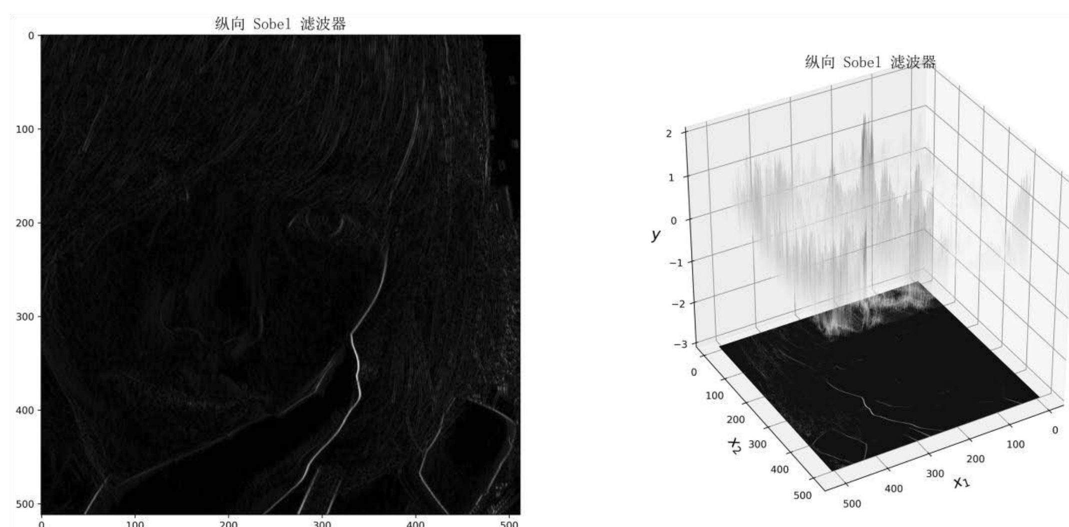


图 9-16 纵向和横向 Sobel 滤波器的效果

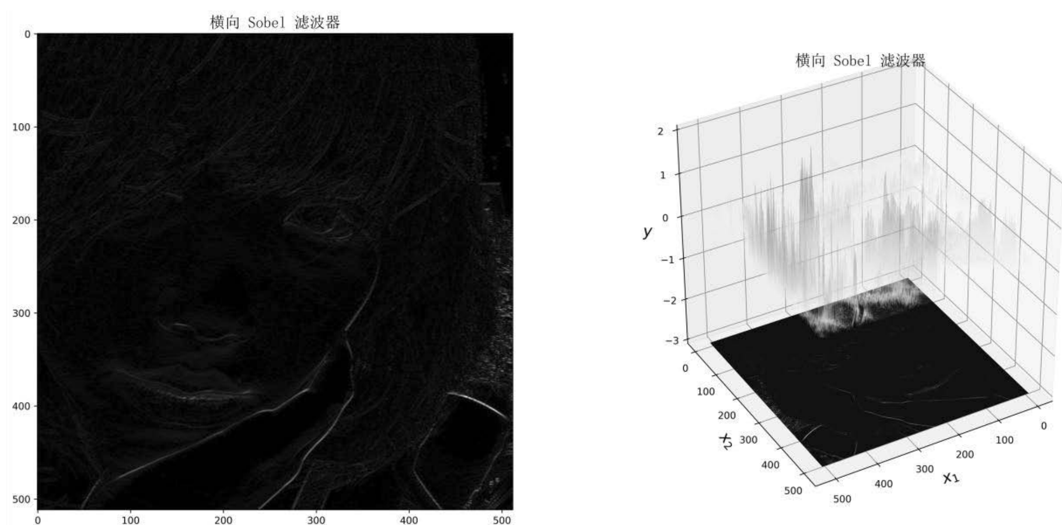


图 9-16　（续）

黑色区域对应原始图像中非边缘的区域，较亮的线条对应原始图像中的边缘。纵向 Sobel 滤波器检测纵向边缘，横向 Sobel 滤波器检测横向边缘，若将两种 Sobel 滤波器的结果（绝对值）相加，则可以检测原始图像中的各向边缘，如图 9-17 所示。

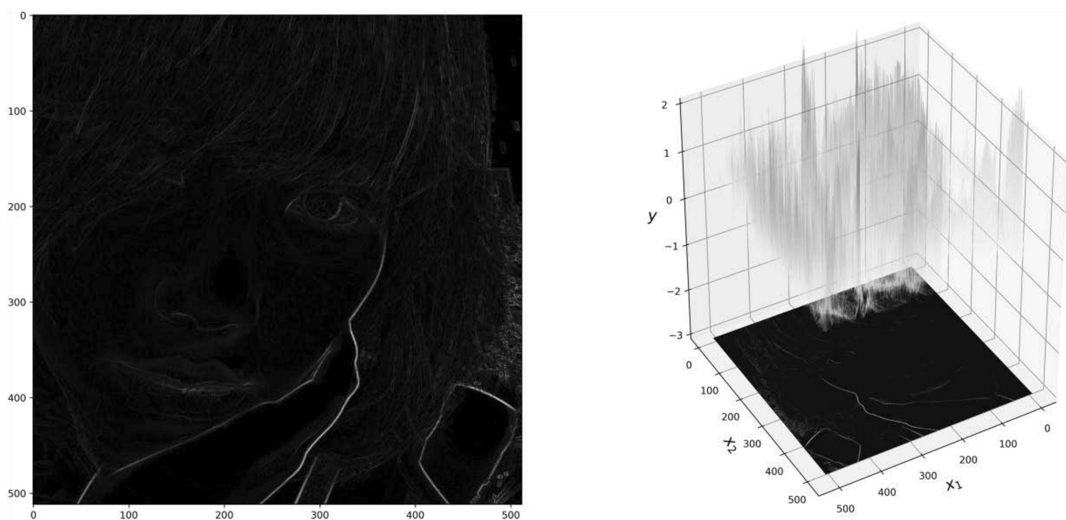


图 9-17　Sobel 滤波器检测图像边缘

在机器视觉领域，滤波器可以用来提取图像特征。举一个简单的例子：问题的目标是判断图像中墙上是否有门，将图像分为无门和有门两类，这是一个二分类问题。我们要构建一个模型，

以图像为输入，输出图像中有门的概率。

我们的模型如图 9-18 所示，它对图像施加纵向和横向 Sobel 滤波器，得到两幅边缘图，求两幅边缘图的所有像素值的平方和，然后施加 Tanh 激活函数。

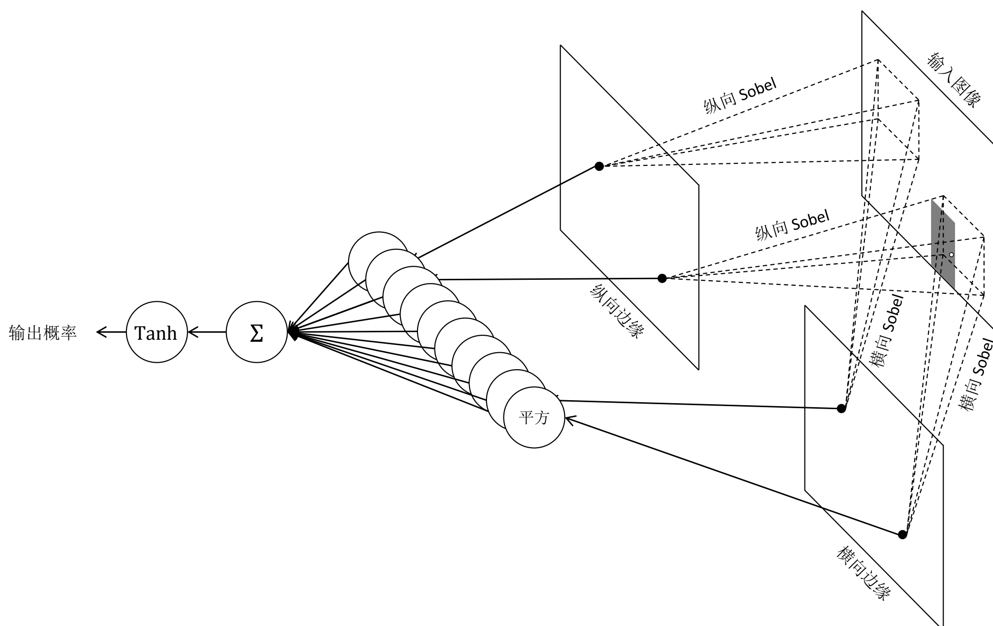


图 9-18　用 Sobel 滤波器提取特征识别墙上的门

墙上无门的图像是一片单色，缺乏边缘。对于无门的图像，两个 Sobel 滤波器的输出的绝对值较小。相反，有门的图像有清晰的边缘，两个 Sobel 滤波器能够检测出这些边缘，它们的部分输出的绝对值较大。将两个 Sobel 滤波器的输出的平方和送给 Tanh 函数，Tanh 的输入在无门时较小，输出接近 0；Tanh 的输入在有门时较大，输出接近 1。把 Tanh 的输出解释为有门的概率，我们就得到了一个识别图像中墙上是否有门的分类器。

这个模型未经验证，但它说明了滤波器的作用。利用 Sobel 滤波器，模型能从图像中提取“边缘”这一特征。不论门的边缘出现在图像的什么位置，模型都能将其检测出来，并反映在 Tanh 的输出中。这个模型其实是一个手工构造的 CNN，两个 Sobel 滤波器构成它的卷积层。

传统上，各种滤波器都是人为有目的地设计出来的。上述例子中，我们知道 Sobel 滤波器的功能，有意用它提取我们认为有用的特征——边缘。如果不由人来设计滤波器，而是从数据中训练滤波器会怎样呢？这就是 CNN 的思想。

9.2　卷积神经网络的组件

CNN 不是指某一特定的网络结构，具有卷积层的神经网络都可以称作 CNN。除卷积层外，CNN 还有其他种类的层，例如激活层、池化层、全连接层等。将这些层以不同规模、不同方式组合叠加，可以构成无穷种不同的 CNN。本节我们介绍 CNN 的各种层和组件。

9.2.1　卷积层

原则上 CNN 可以接受任意维度的输入，因为卷积可以施加于任意函数，但以图像为输入最为常见，所以后文只讨论接受图像的 CNN。但是要注意，图像数据通常是三维的：除了图像的宽与高之外，还有通道（channel）。例如 RGB 彩色图像包含三个通道，对应红、绿和蓝三种颜色。每个通道是二维矩阵，矩阵元素是特定位置上相应颜色的灰度值。所以宽为w、高为h的 RGB 彩色图像其实是$w \times h \times 3$的数据阵列。黑白图像是单通道的，它是$w \times h \times 1$的数据阵列。CNN 可以接受任意通道数量的图像，3 通道最常见。

卷积层对输入图像的多个通道一起施加离散卷积运算。如果卷积核的尺寸是$n \times n$且输入图像有d个通道，则卷积核其实是$n \times n \times d$的数据阵列。以卷积核为权重，对图像以(x, y)为中心的$n \times n$区域内d个通道共$n \times n \times d$个值加权求和再加偏置，就得到(x, y)位置的输出。对输入图像的所有位置执行这种操作，就得到该卷积核的输出图像，如图 9-19 所示。

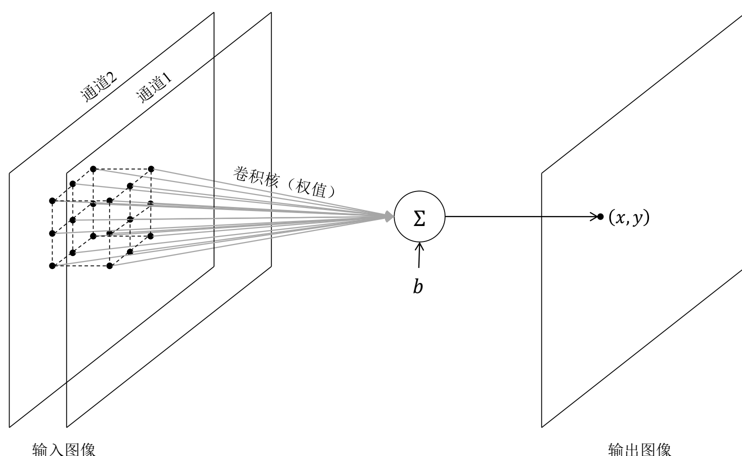


图 9-19　CNN 的卷积计算

卷积核的输出是一个单通道图像，称为一个特征图（feature map）。注意，特征图的值不一定是[0, 255]区间内的整数，不一定是合法的灰度值。称它们为“图像”只是由于数据的空间排列。

求卷积时，输入图像边缘的四周有可能缺失，常用的填充方式有如下 4 种。

- 舍弃（valid）：抛弃边缘位置；
- 补零（same）：用 0 值填充缺失的值；
- 翻转（wrap）：类似吃豆人游戏，若越过边界则从对侧进入图像；
- 镜像（mirror）：按边界取镜像，以镜像对侧的值填充缺失值。

“舍弃”与其他方式不同，它会导致输出特征图的尺寸缩小。特征图的尺寸还与另一个设计选择——步幅（stride）有关。CNN 允许卷积操作不对图像的每个位置都执行，而是每“步幅”个位置（即每隔“步幅减 1”个位置）执行一次。可分别设置横向和纵向步幅，步幅影响特征图的尺寸，若步幅为 2，则每隔一个位置计算一个输出，特征图的尺寸缩小为输入图像的一半。步幅为 1 或 2 的情况如图 9-20 所示。

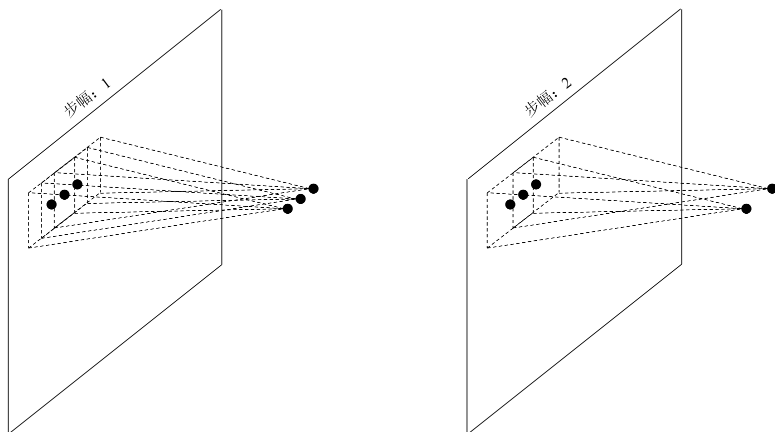


图 9-20　步幅为 1 与步幅为 2

一个卷积层可以有多个卷积核，每个卷积核都能得到一个特征图。CNN 要求同一卷积层的多个卷积核的尺寸、填充方式和步幅都相同，所以它们的特征图尺寸相同，叠加在一起可认为是一幅图像的多个通道，这幅多通道图像就是该卷积层的输出图像。

卷积层有多少卷积核就有多少特征图，每个特征图都是输出图像的一个通道。卷积层的卷积核数、特征图数和输出图像的通道数是同一个数。后文在谈到卷积层的配置时使用“卷积核数”，谈到卷积层的输出时使用“通道数”或“特征图数”，它们指的都是同一个数量。

举一个例子，输入是 $500 \times 400 \times 3$ 的图像，卷积层包含 32 个卷积核，卷积核尺寸是 3×3，补零填充，横向和纵向步幅都为 2，则该卷积层的参数个数是 $32 \times (3 \times 3 \times 3 + 1)$。加上 1 是因

为每个卷积核带一个偏置。

因为横向和纵向步幅都为 2，每隔一个像素计算一个输出，所以特征图的宽和高都是输入图像的一半，其尺寸是 250×200。因为有 32 个卷积核，所以该卷积层输出 32 个特征图，将它们视为 32 个通道，该卷积层的输出就是 $250 \times 200 \times 32$ 的图像。

卷积层不是全连接的，它根据输入的空间位置形成一簇簇连接，并且这些连接簇共享一套权值，即卷积核，这称为权值共享（weight sharing）。权值共享大大减少了参数数量。以上述例子为例，该卷积层有 $250 \times 200 \times 32$（$= 1.6 \times 10^6$）个神经元，有 $500 \times 400 \times 3$ 个输入，若是全连接的话将有 $250 \times 200 \times 32 \times 500 \times 400 \times 3 + 250 \times 200 \times 32$（$\approx 9.6 \times 10^{11}$）个参数，而参数共享的卷积层只有 896 个参数。

卷积层的卷积核就是滤波器，这些滤波器并非人为设计，而是训练而得。可以观察一个训练好的卷积层的各个卷积核，有时能看出它们的功能，例如边缘检测等，下一章中我们会看到这样的例子。卷积层的整体示意图如图 9-21 所示。

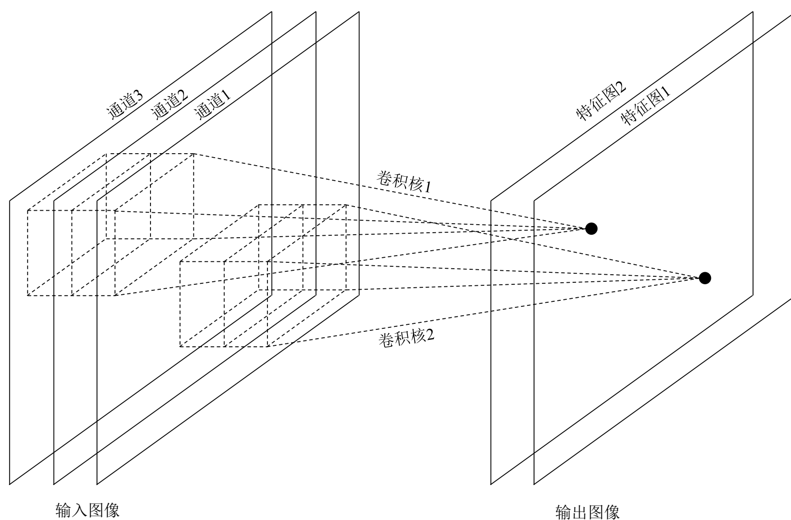


图 9-21　卷积层示意图

9.2.2　激活层

激活层（activation layer）其实就是对上一层神经元的输出施加激活函数。可以选择前文介绍的任何一种激活函数，但在现代深度学习领域，ReLU 及其变体最为常用。卷积层的神经元执行的是仿射函数，与激活层合起来相当于一个传统神经元。激活层如图 9-22 所示。

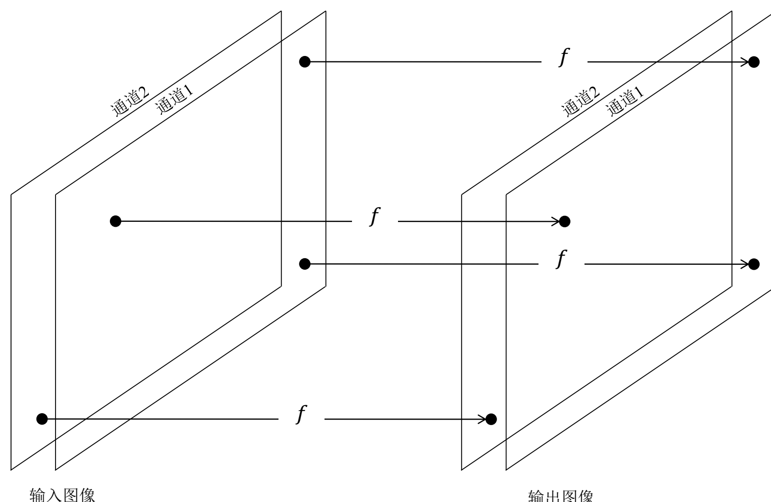


图 9-22　激活层示意图

9.2.3　池化层

池化层（polling layer）用小区域覆盖输入图像的每个通道，从每个小区域计算出一个值，这个小区域称作池化窗口。池化层也有尺寸、填充方式和步幅三个设计选择。尺寸就是池化窗口的大小，填充方式和步幅的含义与卷积层一样。池化层从以输入图像每个像素为中心的池化窗口中计算出一个值，计算方式有两种。

- 最大值池化（max pooling）：选择池化窗口中的最大值；
- 平均值池化（average pooling）：计算池化窗口中所有值的平均值。

延续之前的例子，在激活层后连接一个最大值池化层。若该池化层的尺寸是 2×2，补零填充，横向和纵向步幅都是 2，则该池化层的输出是一幅 $125\times100\times32$ 的图像。池化运算对图像的每个通道分别进行，故输出图像的通道数不变。

池化层没有参数，但是作为计算图的一个节点，它需要计算对每一个输入值的偏导数。令某个池化层神经元的输入是$x_1, x_2, \cdots, x_n$，平均值池化的计算是：

$$f(x_1, x_2, \cdots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (9.10)$$

它对x_j的偏导数是：

$$\frac{\partial f(x_1, x_2, \cdots, x_n)}{\partial x_j} = \frac{1}{n} \qquad (9.11)$$

9

最大值池化的计算是：

$$f(x_1, x_2, \cdots, x_n) = \max(x_1, x_2, \cdots, x_n) \tag{9.12}$$

max不是一个连续函数，如何求偏导数？当$x_1, x_2, \cdots, x_n$都大于 0 时，可以用一个连续函数来近似max：

$$\max(x_1, x_2, \cdots, x_n) \approx \tilde{f}(x_1, x_2, \cdots, x_n) = \frac{x_1^{t+1}}{\sum x_i^t} + \frac{x_2^{t+1}}{\sum x_i^t} + \cdots + \frac{x_n^{t+1}}{\sum x_i^t} = \frac{\sum x_i^{t+1}}{\sum x_i^t} \tag{9.13}$$

注意式（9.13）的第j项：

$$\frac{x_j^{t+1}}{\sum x_i^t} = \frac{x_j}{\sum\left(\frac{x_i}{x_j}\right)^t} \tag{9.14}$$

假设$x_1, x_2, \cdots, x_n$中有k个值达到max，如果x_j小于max，当t趋近于无穷时，式（9.14）趋近于 0；如果x_j等于max，则分母中有k个项为 1，其余项小于 1，当t趋近于无穷时，小于 1 的项都趋近于 0，式（9.14）趋近于$\frac{x_j}{k}$。

所以，当t趋近于无穷时，式（9.13）中有k个项趋近于$\frac{\max}{k}$，其他项都趋近于 0，即式（9.13）趋近于max。当t非常大时，式（9.13）能够很好地模拟max函数。现在我们求$\tilde{f}(x_1, x_2, \cdots, x_n)$对$x_j$的偏导数：

$$\frac{\partial \tilde{f}(x_1, x_2, \cdots, x_n)}{\partial x_j} = \frac{(t+1)x_j^t \cdot \sum x_i^t - t x_j^{t-1} \cdot \sum x_i^{t+1}}{\left(\sum x_i^t\right)^2} = \frac{x_j^t}{\sum x_i^t} + t\frac{x_j^t}{\sum x_i^t}\left(1 - \frac{1}{x_j} \cdot \frac{\sum x_i^{t+1}}{\sum x_i^t}\right) \tag{9.15}$$

式（9.15）最右的第一项是：

$$\frac{x_j^t}{\sum x_i^t} = \frac{1}{\sum\left(\frac{x_i}{x_j}\right)^t} \tag{9.16}$$

当t趋近于无穷时，如果x_j等于max，式（9.16）趋近于$\frac{1}{k}$，否则趋近于 0。考察式（9.15）最右第二项的括号中的部分，若x_j等于max，其极限是 0，否则其极限是有穷值。考察括号外的因子：

$$t\frac{x_j^t}{\sum x_i^t} = \frac{t}{\sum\left(\frac{x_i}{x_j}\right)^t} \tag{9.17}$$

欲求式（9.17）的极限，运用洛必达法则，对分子和分母求导：

$$\lim_{t\to\infty} t\frac{x_j^t}{\sum x_i^t} = \lim_{t\to\infty} \frac{t}{\sum\left(\frac{x_i}{x_j}\right)^t} = \lim_{t\to\infty} \frac{1}{\sum\left(\frac{x_i}{x_j}\right)^t \cdot \log\frac{x_i}{x_j}} \tag{9.18}$$

若x_j等于max，该极限有穷，否则该极限为 0。所以当t趋近于无穷时，式（9.15）最右第二项趋近于 0。

综合起来，当t趋近于无穷时，若x_j等于max，式（9.15）趋近于$\frac{1}{k}$，否则趋近于 0。也就是说当t非常大时，若x_j等于max，则$\tilde{f}(x_1, x_2, \cdots, x_n)$对$x_j$的偏导数近似$\frac{1}{k}$，否则近似为 0。于是最大值池化对输入的偏导可以取：

$$\frac{\partial \max(x_1, x_2, \cdots, x_n)}{\partial x_j} = \begin{cases} \frac{1}{k}, & x_j = \max(x_1, x_2, \cdots, x_n) \\ 0, & x_j < \max(x_1, x_2, \cdots, x_n) \end{cases} \tag{9.19}$$

注意，刚刚的讨论要求所有输入值都大于 0，当输入值可正可负时，可以认为max函数近似于复合函数$\log\tilde{f}(\mathrm{e}^{x_1}, \mathrm{e}^{x_2}, \cdots, \mathrm{e}^{x_n})$，其中$\tilde{f}$的输入都是正值。根据链式法则，若$x_j$是最大值，则偏导数近似为$\frac{1}{\mathrm{e}^{\max}} \cdot \frac{1}{k} \cdot \mathrm{e}^{\max} = \frac{1}{k}$；若$x_j$不是最大值，则偏导数近似为$\frac{1}{\mathrm{e}^{\max}} \cdot 0 \cdot \mathrm{e}^{\max} = 0$。最大值池化层如图 9-23 所示。

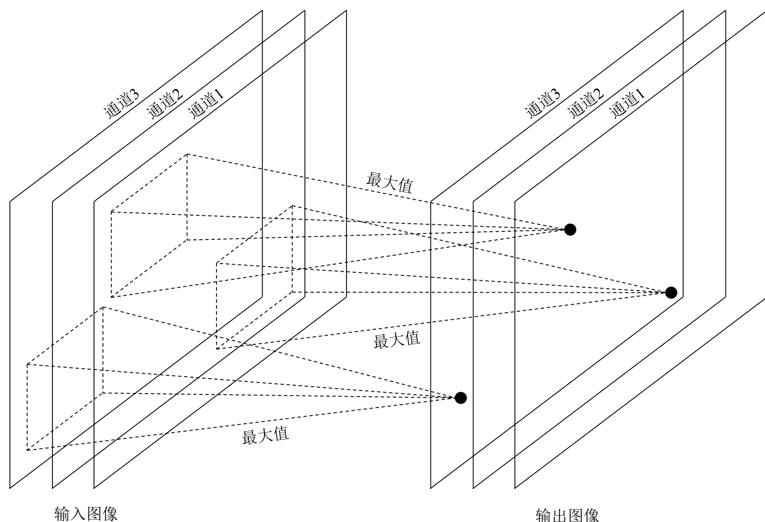


图 9-23　最大值池化层示意图

9.2.4　全连接层

全连接层无视输入数据的空间结构，它的每一个神经元与每一个输入连接，对它们施加仿射函数加激活函数。全连接层一般位于 CNN 的最后，典型的 CNN 交替叠加若干卷积层（包括激活层）和池化层，最后破坏数据的空间结构，将数据展开为向量，连接到全连接层。

前文提到，卷积层可以视作特征提取器，多卷积层相当于进行多轮特征提取，在此过程中压

缩图像尺寸，增加通道数，逐渐提高特征的抽象程度，最终形成特征向量，提交给全连接层。全连接层如图 9-24 所示。

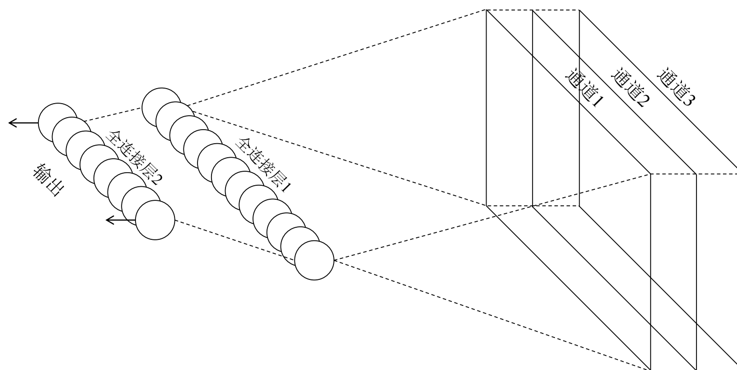


图 9-24　全连接层示意图

9.2.5　跳跃连接

跳跃连接（skip connection）又称短路连接（shortcut connection），它跨过若干层，将输入直接与这几层最终的输出相加，如图 9-25 所示。

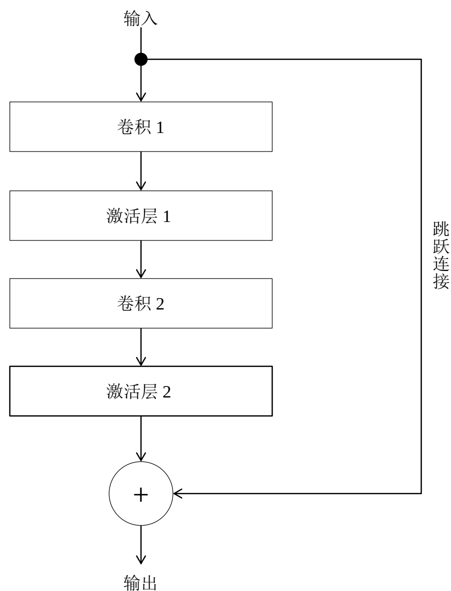


图 9-25　跳跃连接示意图

要使相加可行，被跨过的层不能改变数据形状，即输出图像的尺寸和通道数必须与输入图像一致。也可以给跳跃连接本身添加一个卷积层，该卷积层调整图像尺寸和通道数，以保证两条路径的数据形状相同，如图 9-26 所示。

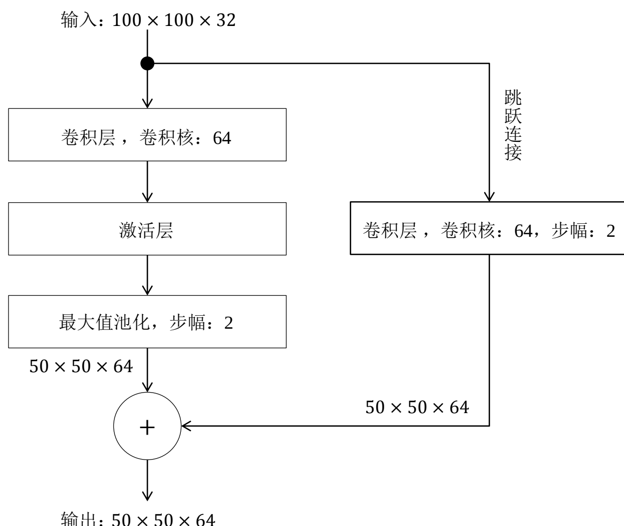


图 9-26　调整图像尺寸和通道数的跳跃连接

被跨过的层称为短路层，短路层学习的是目标与输入的差——残差（residual），这称为残差学习（residual learning）。神经网络的参数一般被初始化为接近 0 的值，短路层初始时的输出接近 0，与输入相加后近似等于输入，所以带跳跃连接的短路层在训练开始时接近恒等映射。深度学习的网络深度较大，中间若干层的局部目标函数通常离恒等映射不远，从恒等映射出发学习局部目标函数，要比从“零函数”出发学习得更快，所以残差学习可以提高训练效率。

假如短路层因为某种原因训练得很慢，跳跃连接可以把短路层之前部分的输出直接传递给短路层之后的部分，于是短路层训练缓慢不会影响网络前后其余部分的训练。残差学习提高了训练深层网络的可行性，残差网络（ResNet）就是一种运用跳跃连接进行残差学习的极深 CNN。

本节介绍了 CNN 的常用组件，一般 CNN 遵循这样的模式：接受输入图像，交替使用卷积层（含激活层）和池化层，利用步幅和卷积核数量逐渐减小图像尺寸并增加通道数，最后将数据展开后连接全连接层。

图 9-27 展示了一种 CNN——VGG16，它的 5 个最大值池化层的步幅都为 2，将图像尺寸压缩一半。5 个池化层隔开 5 组卷积层，每组的卷积层数量分别为：2、2、3、3、3，每组卷积层最

终的输出尺寸和通道数标记在图中。最后一个池化层后面连接三个全连接层，神经元数分别为：4096、4096 和 1000，最后施加 SoftMax 函数。VGG16 中带参数的层（卷积层和全连接层）共有 16 层，这是它的名称的来历。VGG16 是一个典型的深度 CNN，下一章还会更详细地介绍它。

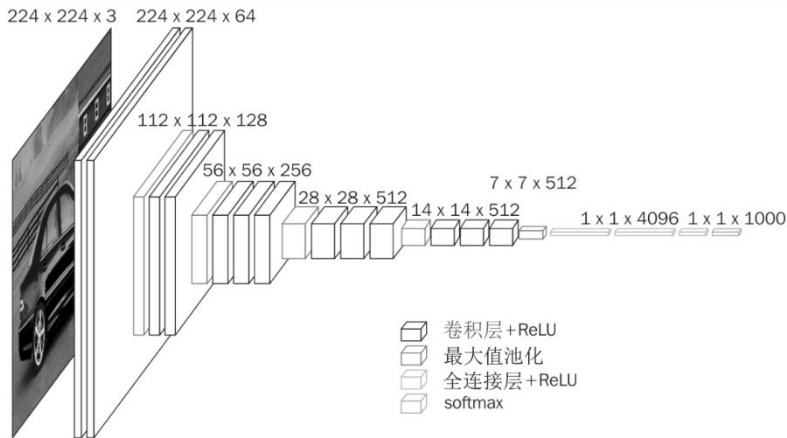


图 9-27　VGG16 示意图

9.3　深度学习的正则化方法

前文介绍过，高自由度的模型容易陷入过拟合，需要对其施以正则化来限制其自由度。CNN 以及其他深度神经网络深度大、参数多，它们的自由度远远高于传统机器学习模型。为了使它们免于过拟合，需要大量的训练数据以及强大的正则化方法。除了传统机器学习和神经网络的正则化方法，例如我们介绍过的$\mathcal{L}_1$和$\mathcal{L}_2$正则化、提前停止等，深度学习还有一些特有的正则化方法，本节介绍其中的几种。

9.3.1　权值衰减

经典的$\mathcal{L}_1$和$\mathcal{L}_2$正则化可以应用于 CNN，一般只对权值施加，不对偏置施加。将 CNN 的全体权值排列成一个向量$\boldsymbol{w}$，构造$\mathcal{L}_2$正则项：

$$\Omega_{\mathcal{L}_2}(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \tag{9.20}$$

或$\mathcal{L}_1$正则项：

$$\Omega_{\mathcal{L}_1}(\boldsymbol{w}) = \lambda\sum_{i=1}^{n}|w_i| \tag{9.21}$$

λ是正则化强度。将正则项加在损失值（例如交叉熵）之上，形成带正则项的损失函数。正像我们介绍过的，$\mathcal{L}_1$和$\mathcal{L}_2$正则化表达了对参数先验分布的假设，最小化带正则项的损失函数相当于最大化参数的后验概率。正则化越强，则参数先验分布的方差越小，最大后验解越接近 0。$\mathcal{L}_1$和$\mathcal{L}_2$正则化倾向于得到绝对值较小的权值。在深度学习领域，它们被称为权值衰减（weight decay）。

9.3.2 Dropout

Dropout 技术很简单，但在实践中有很好的效果。它的做法是：在梯度下降的每一次迭代时，将每个神经元按概率p从网络中去掉。若某神经元在一次迭代时命中概率p，则该神经元不参与前向传播和反向传播及梯度下降。概率p称为 Dropout 率（Dropout rate）。

Dropout 阻止某个神经元成为关键节点。训练时任何一个神经元都有可能缺席，这迫使其他神经元承担该缺席神经元的责任，将责任分散。神经元的输入是其他神经元的输出，它们有可能缺席，于是神经元不会过重地依赖某个特定输入。

有一个重要的技术细节是，如果在训练时每个神经元以概率p缺席，那么统计意义上每个神经元训练时的输入数量是预测时的$1-p$倍，神经元在预测时接受的信号强度是训练时的$1/(1-p)$。为纠正这种偏差，可在训练完成后将所有权值乘上$1-p$，将预测时的信号强度拉回训练时的水平。另一种办法是训练时将每个神经元的输出乘以$1/(1-p)$，放大输出以保持训练时和预测时信号强度相同。这两种办法并不等价，但是效果相似。

可以从模型集成的角度来解释 Dropout 的作用。模型集成有许多形式，最简单的一种是，训练多个模型，预测时取它们结果的平均。模型集成的作用是降低方差、防止过拟合。一般而言，参与集成的模型越独立越好，可以采用有放回采样，为每一个参与集成的模型构造独特的训练集，这种方式称为 BAGGING（bootstrap aggregating）。

假设神经网络有N个神经元，每个神经元有参与或缺席两种情况，则可以形成2^N种不同的神经网络。每一次迭代都相当于用一部分样本（一个 Mini Batch）训练其中一种神经网络，进行K次迭代就训练了K个神经网络。预测时全体神经元都参与，相当于对K个神经网络取平均，这可以视为 BAGGING 过程。Dropout 率越大，每次迭代缺席的神经元越多，各神经网络之间的差别越大。所以，提高 Dropout 率起到的作用是降低方差、防止过拟合。

9.3.3 权值初始化

在初始化权值时，最好保证各个激活层神经元的输出值（简称“激活值”）有较好的分布。如果采用双侧饱和的激活函数，例如 Logistic，要保证激活值不能集中于 0 和 1，否则大部分神

经元处于饱和区域，梯度过小。激活值的方差应该较大，这样初始时神经元之间有较大差异性，可防止它们协同变化。研究证明：引入神经元所在层的输入连接数n_{in}（扇入）和输出连接数n_{out}（扇出），并以特定方式初始化权值和偏置，可保证激活值有较好的分布，本书省略证明。“Xavier 初始化”以正态分布$\mathcal{N}(0,\sigma^2)$初始化某层的权值，其中：

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}} \tag{9.22}$$

或者以$[-r,r]$上的均匀分布初始化某层的权值，其中：

$$r = \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}} \tag{9.23}$$

“Xavier 初始化”适用于 Logistic 这种在原点附近接近线性，在正负两侧饱和的激活函数。对 ReLU 及其变体可采用“He 初始化”，它以正态分布$\mathcal{N}(0,\sigma^2)$初始化某层的权值，其中：

$$\sigma = \sqrt{\frac{4}{n_{\text{in}}+n_{\text{out}}}} \tag{9.24}$$

或者以$[-r,r]$上的均匀分布初始化某层的权值，其中：

$$r = \sqrt{\frac{12}{n_{\text{in}}+n_{\text{out}}}} \tag{9.25}$$

9.3.4　批标准化

在深度神经网络的训练过程中，网络后部的层的输入分布会因前面层的权值和偏置的更新而持续发生变化，后部的层需要不停适应输入分布的变化，造成训练困难，这称为内部协变量漂移（internal covariate shift，ICS）。批标准化（batch normalization，BN）是一种对抗 ICS 的技术。可以将 BN 视为一层，插在激活层之前。BN 将激活函数的输入值进行标准化，然后再次平移和缩放。

训练时，某个激活层神经元在一个 Mini Batch 中得到M个输入值：a_i（$i=1,\cdots,M$），M是 Mini Batch 的样本数量，BN 首先将这些输入值标准化：

$$\widehat{a_i} = \frac{a_i-\mu_b}{\sqrt{\sigma_b^2+\epsilon}}，\quad i=1,\cdots,M \tag{9.26}$$

ϵ是防止分母为 0 的一个极小值。μ_b是一个 Mini Batch 的所有输入值的均值：

$$\mu_b = \frac{1}{M}\sum_{i=1}^{M} a_i \tag{9.27}$$

σ_b^2是一个 Mini Batch 的所有输入值的样本方差：

$$\sigma_b^2 = \frac{1}{M-1}\sum_{i=1}^{M}(a_i - \mu_b)^2 \tag{9.28}$$

标准化后，BN 再用γ和β两个参数将$\widehat{a_i}$（$i = 1,\cdots,M$）进行平移和缩放：

$$z_i = \gamma\widehat{a_i} + \beta, \quad i = 1,\cdots,M \tag{9.29}$$

最终以z_i（$i = 1,\cdots,M$）作为该激活层神经元的输入。BN 有 4 个参数μ_b、σ_b^2、γ和β。γ和β参与训练。训练时，μ_b和σ_b^2对每一个 Mini Batch 计算。预测时使用的均值和样本方差可在训练时用滑动平均进行估计：

$$\mu = (1-\alpha)\mu + \alpha\mu_b \tag{9.30}$$

以及

$$\sigma^2 = (1-\alpha)\sigma^2 + \alpha\sigma_b^2 \tag{9.31}$$

$0 < \alpha < 1$是遗忘因子。

9.3.5 数据增强

再强大的正则化方法也弥补不了数据的缺失，深度神经网络是机器学习领域迄今最复杂、自由度最高的模型，它需要大量训练样本。数据增强（data augment）是一种简单而有效的正则化技术。简单来说，数据增强就是用已有的样本生成新的样本。对于图像来说，数据增强就是用已有的图像生成新的图像。生成新图像的操作包括剪裁、平移、旋转、镜像翻转、调整亮度及对比度等，或同时施加多个操作。要保证变化后的图像与原图像仍属于同一个类别，即变化不影响图像内容的本质。

新图像都来源于已有的图像，某种意义上数据增强并没有提供新的信息，但是图像的各种变化引入了不影响图像内容的噪声，例如，旋转、调整对比度或镜像翻转都不改变图像中物体的类别，这可以避免模型受到非本质特征的干扰，防止过拟合。

9.4 小结

卷积神经网络得名于卷积，本章首先简述了连续和离散卷积的原理，并从滤波器角度介绍了卷积在数字图像处理领域的应用。在前深度学习时代，人们使用各种人为设计的滤波器从图像中提取特征，再将特征提交给传统机器学习模型。

可以认为 CNN 的卷积层是可训练的滤波器，它们在训练中习得某种有用的功能，成为特征提取器。卷积层、激活层、池化层交替叠加，逐步提取更高抽象层次的特征，最终将特征提交给全连接层。全连接层就相当于以网络前部提取的特征向量为输入的全连接神经网络，这就是典型的 CNN 结构。

无论多么复杂的 CNN，它仍然是一个计算图，可以运用计算图自动求导加梯度下降来训练。有了前文的铺垫，读者已经能够理解任何 CNN 的训练算法。本章还介绍了一些深度学习特有的正则化技术，例如 Dropout、权值初始化、批标准化、数据增强等。下一章我们将介绍几种经典的 CNN。

第 10 章

经典 CNN

10

上一章我们介绍了 CNN 的原理及其各种组件，真实世界的 CNN 都是这些组件的组合。本章按时间顺序介绍 5 种经典的 CNN：LeNet-5、AlexNet、VGGNet、GoogLeNet 和 ResNet。这 5 种 CNN 各具特色，除了深度不断加深外，还引入了如 Inception 模块和残差单元这样的新结构。

这几种 CNN（除了 LeNet-5）在 2012~2015 年的 ILSVRC 竞赛中不断创造新的好成绩。若与物种的演化进行类比，则竞赛可以看作一种选择压力，驱使人们尝试各种新的网络结构和技术。在竞赛中表现优异的网络获得生存优势，并在现有基础上分化、变形、发展，犹如生物的演化。本章我们尝试像古生物学家那样，从极有限的标本中勾勒 CNN 的演化趋势，分析使它们的表现得到提升的结构特点。

10.1　LeNet-5

LeNet-5 是 Yann LeCun、Leon Bottou、Yoshua Bengio 和 Patrick Haffner 于 1998 年的论文 *Gradient Based Learning Applied to Document Recognition* [1]中提出的一种早期 CNN 结构，它被用于 MNIST 数据集手写数字识别。

MNIST 是美国国家标准与技术研究所发布的手写数字图像数据集。该数据集分成 10 个类别：数字 0~9。训练集包含 60 000 个带类别标签的样本，测试集包含 10 000 个不带标签的样本。样本是 28×28 单通道灰度图像，像素值位于[0, 255]区间，代表该像素的灰度。图 10-1 展示了 MNIST 数据集的例子。

LeNet-5 的输入不是原始 MNIST 图像。首先将图像四周以零值扩充两个像素的宽度，使图像尺寸变成 32×32，这是为了使手写数字的笔画都能位于某个卷积区域的中心。之后再将元素值除以 255，归一化到[0, 1]。因为图像是单通道的，所以 LeNet-5 的输入数据的形状是 $32 \times 32 \times 1$，值为[0, 1]内的实数。



图 10-1　MNIST 手写数字图像样例

LeNet-5 的结构如表 10-1 所示。表中包含各层的名称/代号、类型、尺寸（卷积核或池化窗口）、步幅、填充方式、输出尺寸、输出通道数以及激活函数。我们不把激活函数当作单独的一层，而是标注在每一层中，即对该层的输出施加某类型的激活函数。

表 10-1　LeNet-5 的结构

层	类　型	尺　寸	步　幅	填充方式	输出尺寸	输出通道数	激活函数
输入层	—	—	—	—	32 × 32	1	—
C1	卷积	5 × 5	1	舍弃	28 × 28	6	Tanh
S2	平均值池化	2 × 2	2	补零	14 × 14	6	Tanh
C3	卷积	5 × 5	1	舍弃	10 × 10	16	Tanh
S4	平均值池化	2 × 2	2	补零	5 × 5	16	Tanh
C5	卷积	5 × 5	1	舍弃	1 × 1	120	Tanh
F6	全连接	—	—	—	84	—	Tanh
输出层	全连接	—	—	—	10	—	grbf

LeNet-5 的结构如图 10-2 所示。

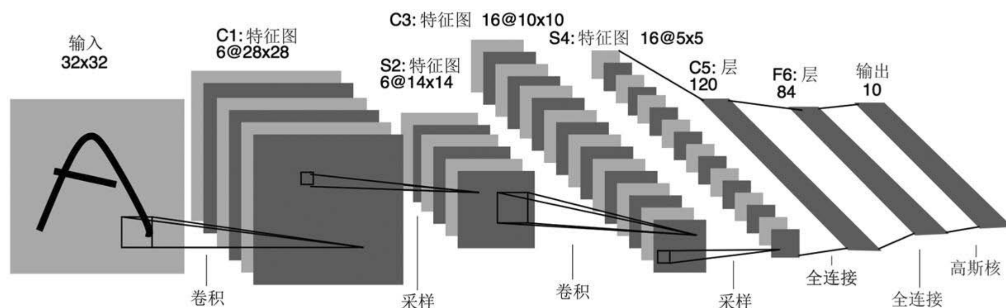


图 10-2　LeNet-5 示意图[1]

LeNet-5 的卷积层采用“舍弃”填充，因为其卷积核尺寸都是 5×5，所以位于上下左右边缘的两个像素被舍弃，输出图像的宽和高比输入图像小 4 个像素。C5 卷积层的输入尺寸是 5×5，它的输出尺寸是 1×1（一个值）。C5 有 120 个卷积核，所以它的输出实际是一个 120 维向量。

LeNet-5 的平均值池化层采用补零填充，步幅为 2，它们将图像尺寸压缩一半。LeNet-5 的平均值池化层与通常的平均值池化层不同：它给平均值乘上一个系数，再加一个偏置。每个输入通道都有独立的系数和偏置，并参与训练。池化层 S2 有 6 个输出通道，卷积层 C3 有 16 个卷积核，但 C3 的每个卷积核并不与 S2 的所有输出通道连接，C3 与 S2 之间的连接关系如表 10-2 所示。

表 10-2 LeNet-5 的卷积层 C3 与池化层 S2 的连接关系

S2/C3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	√				√	√	√			√	√	√	√		√	√
1	√	√				√	√	√			√	√	√	√		√
2	√	√	√				√	√	√			√		√	√	√
3		√	√	√			√	√	√	√			√		√	√
4			√	√	√			√	√	√	√		√	√		√
5				√	√	√			√	√	√	√		√	√	√

从表 10-2 可见，C3 前 6 个卷积核各连接 S2 的 3 个输出通道，它们的形状是 5×5×3。后面 9 个卷积核各连接 S2 的 4 个输出通道，它们的形状是 5×5×4。最后一个卷积核连接 S2 的全部 6 个输出通道，它的形状是 5×5×6。

LeNet-5 的输出层有 10 个神经元，F6 层有 84 个神经元，输出层与 F6 层全连接，所以输出层每个神经元的权值向量是 84 维向量。图 10-3 展示了部分 ASCII 可见字符的 12×7 原型（prototype）。

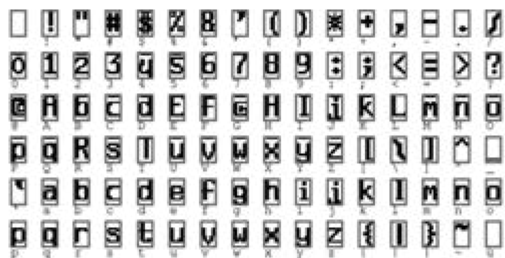


图 10-3 部分 ASCII 可见字符的 12×7 原型

原型的元素取值−1 或+1，表示背景或前景。LeNet-5 用数字 0~9 的原型作为输出层神经元权值向量的初始值。输出层神经元的权值向量既然是数字原型，就可以用 F6 层的输出向量与这些原型向量比较。记 F6 层的输出向量为$\boldsymbol{x}$,（从 0 开始）第j个输出层神经元的权值向量是$\boldsymbol{w}^j$，该神

经元执行的计算是：

$$\mathrm{grbf}(\boldsymbol{x},\boldsymbol{w}^j)=\left\|\boldsymbol{x}-\boldsymbol{w}^j\right\|^2=\sum_{i=1}^{n}\left(x_i-w_i^j\right)^2\text{，}\quad j=0,\cdots,9 \tag{10.1}$$

式（10.1）称为高斯径向基函数。高斯径向基是径向基函数（radial basis function）的一种。径向基函数的输出只取决于输入向量与原型之间的距离。可以取不同的距离定义，高斯径向基取欧氏距离的平方，即输入向量与原型各分量之差的平方和。输出层 10 个神经元分别代表数字 0~9，取输出值最小的神经元所代表的数字为预测结果，因为这个神经元的权值向量（原型）与样本的 F6 输出最接近。最小化式（10.1）等价于最大化：

$$p(\boldsymbol{x}|\boldsymbol{w}^j)=\frac{1}{(2\pi\sigma^2)^{n/2}}\mathrm{e}^{-\frac{\left\|\boldsymbol{x}-\boldsymbol{w}^j\right\|^2}{2\sigma^2}}\text{，}\quad j=0,\cdots,9 \tag{10.2}$$

使式（10.1）最小的$\boldsymbol{w}^j$使似然概率$p(\boldsymbol{x}|\boldsymbol{w}^j)$最大，所以它是最大似然解。输出层权值向量可以参与训练，也可以不参与训练。令训练集为$\{\boldsymbol{X}^i,y^i\}_{i=1}^{M}$，每个$\boldsymbol{X}^i$是手写数字图像，是 32 × 32 × 1 的阵列。每个y^i取值 0~9，是样本的真实类别。LeNet-5 有 10 个输出：

$$z^j(\boldsymbol{X}^i,\boldsymbol{W})=\mathrm{grbf}(\boldsymbol{x}^i,\boldsymbol{w}^j)\text{，}\quad j=0,\cdots,9 \tag{10.3}$$

其中，$\boldsymbol{x}^i$是样本$\boldsymbol{X}^i$的 F6 输出，$\boldsymbol{W}$是网络的全体参数，$z^j(\boldsymbol{X}^i,\boldsymbol{W})$衡量$\boldsymbol{x}^i$与数字$j$的原型$\boldsymbol{w}^j$之间的距离。样本的真实类别是$y^i$，所以训练过程应该最小化所有样本的$z^{y^i}(\boldsymbol{X}^i,\boldsymbol{W})$（$i=1,\cdots,M$），于是可以将损失函数定义为：

$$\mathrm{loss}(\boldsymbol{W})=\frac{1}{M}\sum_{i=1}^{M}z^{y^i}(\boldsymbol{X}^i,\boldsymbol{W})=\frac{1}{M}\sum_{i=1}^{M}\mathrm{grbf}\left(\boldsymbol{x}^i,\boldsymbol{w}^{y^i}\right) \tag{10.4}$$

但如果输出层神经元的权值向量也参与训练，则这个损失函数就存在问题。因为训练过程可以把 10 个权值向量训练成同一个向量，并使 F6 层对所有训练样本都输出该向量，这样的话训练集上的损失可以降为零，但很显然这不是我们想要的结果。在拉近样本的 F6 输出与正确原型之间的距离的同时，应该加大它与错误原型之间的距离。出于这个考虑，给每个样本的损失添加一项：

$$\mathrm{loss}(\boldsymbol{W})=\frac{1}{M}\sum_{i=1}^{M}\left(z^{y^i}(\boldsymbol{X}^i,\boldsymbol{W})+\log\left(\mathrm{e}^{-k}+\sum_{j=0}^{9}\mathrm{e}^{-z^j(\boldsymbol{X}^i,\boldsymbol{W})}\right)\right) \tag{10.5}$$

新添的惩罚项驱使训练过程增大 F6 输出与所有原型之间的距离，旧有的惩罚项拉近 F6 输出与正确原型之间的距离。常数项e^{-k}防止式（10.5）无限降低。有了损失函数，运用计算图反向传播加梯度下降就可以训练 LeNet-5 了。

LeNet-5 是最早的 CNN 之一，它与现今的惯常做法有一些不同：S2 与 C3 之间的局部连接已

无必要，Tanh 激活函数也已被 ReLU 取代。现今，人们会将 F6 层连接一个 10 神经元的全连接层，之后再施加 SoftMax 函数，以交叉熵作为损失值。LeNet-5 有 7 层，以今天的标准看来算不上“深度”，更主要的是，它没有使用深度学习特有的训练和正则化技术，所以它还不算是深度网络。

10.2　AlexNet

AlexNet 是 Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E. Hinton 于 2012 年的论文 *ImageNet Classification with Deep Convolutional Neural Networks*[2]中提出的 CNN 结构。AlexNet 得名自第一作者 Alex Krizhevsky。论文标题出现了“Deep”字样，AlexNet 是一个真正的深度 CNN。

AlexNet 被用于 ImageNet 图像数据集。ImageNet 数据集包含超过 1500 万张带类别标签的高分辨率彩色图像，分属大约 22 000 个类别，采集自网络并经过人工标注。每年的 ILSVRC 竞赛取 ImageNet 数据集的 1000 个类别，共包含 120 万张训练集图像、5 万张验证集图像和 15 万张测试集图像。论文中采用的训练集和测试集来自 ILSVRC-2010。作者以该成果参加了 ILSVRC-2012 竞赛，并以较大优势夺得冠军。图 10-4 展示了一些 ImageNet 图像。



图 10-4　ImageNet 数据集图像样例

ImageNet 图像是大小不一的 RGB 三通道彩色图像。作者首先将每幅图像较短的一边缩放成 224 像素，再从图像中心剪裁出 224 × 224 的区域，最后，将图像每一个像素减去其均值，以去中心化。AlexNet 的输入是 224 × 224 × 3 的阵列，每个元素是去中心化的像素值。AlexNet 的结构如表 10-3 所示。

表 10-3　AlexNet 的结构

层	类　型	尺　寸	填充方式	步　幅	输出尺寸	输出通道数	激活函数
输入层	—	—	—	—	—	3（RGB）	—
C1	卷积	11 × 11	补零	4	55 × 55	96	ReLU
S2	最大值池化	3 × 3	—	2	27 × 27	96	—
C3	卷积	5 × 5	补零	1	27 × 27	256	ReLU
S4	最大值池化	3 × 3	—	2	13 × 13	256	—
C5	卷积	3 × 3	补零	1	13 × 13	384	ReLU
C6	卷积	3 × 3	补零	1	13 × 13	384	ReLU
C7	卷积	3 × 3	补零	1	13 × 13	256	ReLU
S8	最大值池化	3 × 3	—	2	6 × 6	256	—
F9	全连接	—	—	—		4096	ReLU
F10	全连接	—	—	—		4096	ReLU
输出层	全连接	—	—	—		1000	SoftMax

卷积层和池化层的输出尺寸取决于输入尺寸、填充方式和步幅，读者可自行验证。通常，池化层窗口无重叠地覆盖输入图像，但 AlexNet 不同，AlexNet 的池化层窗口的尺寸是 3 × 3，步幅是 2，窗口之间有一个像素的重叠。AlexNet 的卷积层采用 ReLU 激活函数，它是较早应用 ReLU 的神经网路。输出层有 1000 个神经元，对它们的输出施加 SoftMax 函数，得到 1000 个类别的概率。

AlexNet 对 C1 和 C3 卷积层应用了一种称为局部响应标准化（local response normalization，LRN）的技术。假设卷积层有N个卷积核，令$a_{x,y}^{i}$（$i = 0,\cdots,N-1$）是N个卷积核在(x,y)位置上的输出，计算：

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k+\alpha\sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}\left(a_{x,y}^{j}\right)^{2}\right)^{\beta}} \tag{10.6}$$

以$b_{x,y}^{i}$（$i = 0,\cdots,N-1$）作为N个特征图在(x,y)位置的值。$b_{x,y}^{i}$引入第i个卷积核的n个相邻卷积核在相同位置的输出，将这些相邻输出的平方和放在分母上，若某个卷积核的输出较大，它

就会抑制相邻特征图的值。这个机制受到生物神经元侧向抑制效应的启发。k、α、β和n是 LRN 的超参数，论文中的取值为$k = 2$、$\alpha = 10^{-4}$、$\beta = 0.75$、$n = 5$。AlexNet 的结构如图 10-5 所示。

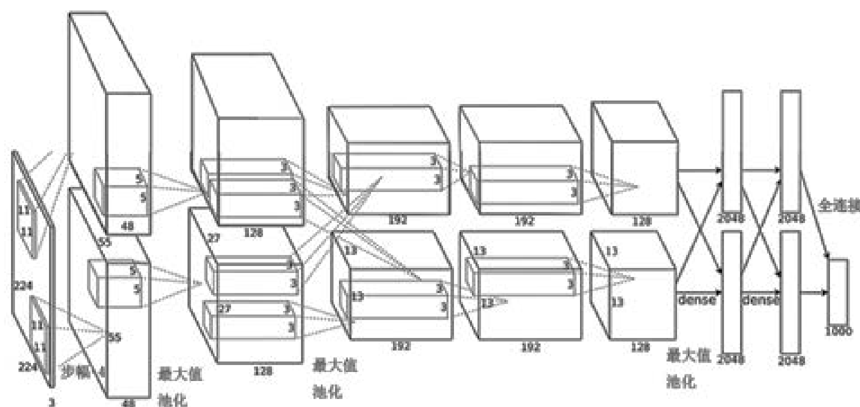


图 10-5　AlexNet 示意图[2]

图中只画了输入、卷积层和全连接层，激活函数作为卷积层的一部分，池化层标注在相应的卷积层之后。注意，图中将每个卷积层的卷积核分成两组，分别画在上部和下部。C3、C6 和 C7 三个卷积层并不连接前一层的全部特征图，而只连接和自己同一组的特征图。作者将两组卷积核分放在两块 GPU 上，这种连接方式减少了 GPU 之间的数据交换，同时也减少了参数量。

Dropout 在当时还是一种新技术，训练时，作者对 F9 和 F10 应用了 Dropout，Dropout 率为 0.5。作者还应用了数据增强，预处理时将图像缩放并裁剪成 256×256 大小，从中随机剪裁出 224×224 的区域，并随机水平翻转。数据增强使训练集扩大了 2048 倍，并有效防止了过拟合。预测时从 256×256 图像的四角和中央剪裁出 5 个 224×224 图像，并对它们水平翻转，共得到 10 幅图像，取 AlexNet 对 10 幅图像预测概率的平均值作为最终预测概率。

作者以 0 均值、0.01 标准差的正态分布初始化各层权值。C3、C6 和 C7 的偏置初始化为 1，其余层的偏置初始化为 0。训练以交叉熵为损失值，一个 Mini Batch 包含 128 个样本，采用冲量梯度下降算法，冲量系数 0.9，施加强度为 0.0005 的$\mathcal{L}_2$正则化。作者采用了一种启发式的学习率调整策略：当观察到验证集错误率不再下降时，手动将学习率设为当前值的 1/10。初始学习率是 0.01，训练结束前共调整了三次，训练进行了 90 个 epoch。

我们来看一下训练完成的 C1 层的 96 个卷积核，这些卷积核都是 $11 \times 11 \times 3$ 的阵列，经过归一化后是三通道彩色图像。图 10-6 将它们展示出来，上半部 48 个卷积核在第一块 GPU 上训练，下半部 48 个卷积核在第二块 GPU 上训练。可看出两组卷积核的功能有所差别，上半部多数卷积

核类似 Sobel 算子，方向各异，它们关注图像中的边缘，下半部多数卷积核三个通道差异明显，它们关注图像的颜色。

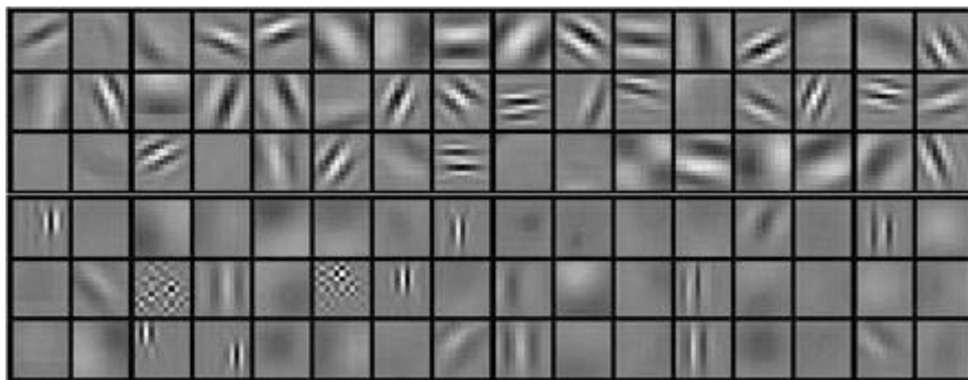


图 10-6　AlexNet 的 C1 层的 96 个卷积核[2]

AlexNet 比 LeNet-5 更深更大，它有超过 65 万个神经元，6 千万个参数。更重要的是，AlexNet 应用了当时最新的 ReLU 激活函数、LRN、Dropout 以及数据增强等技术，它已经跨入了深度学习领域。

10.3　VGGNet

Karen Simonyan 和 Andrew Zisserman 于 2015 年的论文 *Very Deep Convolutional Networks for Large-Scale Image Recognition*[3]中提出并实验了一组 CNN，称它们为 VGGNet（来自作者单位 Visual Geometry Group，Oxford University）。

VGGNet 是一类简单的 CNN，它们包含 5 组卷积层，组间插入池化层以缩小图像尺寸。网络后端连接三个全连接层，神经元数分别为：4096、4096 和 1000。最后施加 SoftMax 函数，得到 1000 类别的概率。卷积层和全连接层的激活函数都是 ReLU。卷积层的填充方式是补零填充，步幅为 1。池化层为最大值池化，采用补零填充，步幅为 2。每经过一个池化层图像尺寸缩小一半。每组卷积层（除最后一组）的卷积核数量扩大 2 倍，即增加通道数。根据组内卷积层数量的不同，作者构造了由浅到深的 6 种 VGGNet，如表 10-4 所示。

表 10-4 6 种 VGGNet 的结构

网络（层数）	A（11）	A-LRN（11）	B（13）	C（16）	D（VGG16）	E（VGG19）
输入	输入层（224×224×3）					
卷积层组 1	3×3,64	3×3,64 LRN	3×3,64 3×3,64	3×3,64 3×3,64	3×3,64 3×3,64	3×3,64 3×3,64
池化层	最大值池化，2×2，步幅 2					
卷积层组 2	3×3,128	3×3,128	3×3,128 3×3,128	3×3,128 3×3,128	3×3,128 3×3,128	3×3,128 3×3,128
池化层	最大值池化，2×2，步幅 2					
卷积层组 3	3×3,256 3×3,256	3×3,256 3×3,256	3×3,256 3×3,256	3×3,256 3×3,256 1×1,256	3×3,256 3×3,256 3×3,256	3×3,256 3×3,256 3×3,256 3×3,256
池化层	最大值池化，2×2，步幅 2					
卷积层组 4	3×3,512 3×3,512	3×3,512 3×3,512	3×3,512 3×3,512	3×3,512 3×3,512 1×1,512	3×3,512 3×3,512 3×3,512	3×3,512 3×3,512 3×3,512 3×3,512
池化层	最大值池化，2×2，步幅 2					
卷积层组 5	3×3,512 3×3,512	3×3,512 3×3,512	3×3,512 3×3,512	3×3,512 3×3,512 1×1,512	3×3,512 3×3,512 3×3,512	3×3,512 3×3,512 3×3,512 3×3,512
池化层	最大值池化，2×2，步幅 2					
全连接层 1	4096					
全连接层 2	4096					
全连接层 3	1000					
SoftMax	1000					

表中层数只计卷积层和全连接层（带参数的层）。$n \times n, m$表示卷积核尺寸为$n \times n$，数量为m。网络 A-LRN 对第一个卷积层的输出施加局部响应标准化（LRN）。VGG 的卷积核尺寸都是 3×3，将多个 3×3 的卷积层叠加，则最后一层的神经元可以感知前面层上更大的区域，如图 10-7 所示。

以叠加小卷积核代替单个大卷积核有两个好处：首先，层之间的非线性激活函数增强了网络的表达能力，其次可以减少参数量。图 10-7 中的连接方式有 9×3 = 27 个权值，而 7×7 的卷积核有 49 个权值。VGG16 的结构如图 10-8 所示。

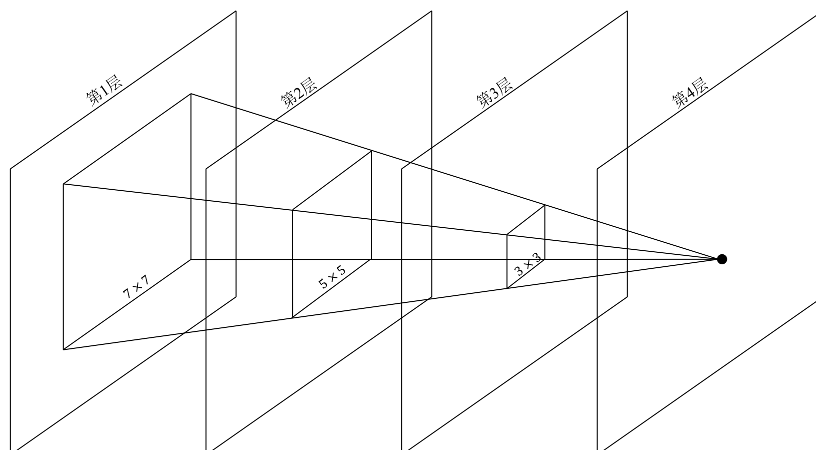


图 10-7　第 4 层的神经元可以感知第 1 层的 7×7 区域

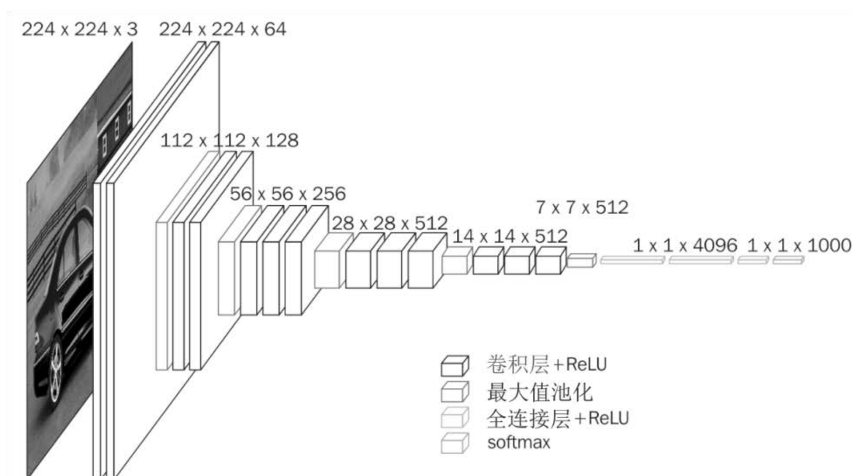


图 10-8　VGG16 示意图

作者首先训练最浅的网络 A，随机初始化权值和偏置，训练完成后，通过添加卷积层将 A 改造成 B，原有的层沿用之前训练好的权值和偏置，新添加的层的权值随机初始化，偏置初始化为零。作者以这种方式依次训练网络 C、D 和 E。

6 种 VGGNet 除了深度逐渐增加，在其他方面都相同，且在同样的条件下训练和测试，实验表明，更大的深度确实对提升 CNN 的表现有作用。加深网络而减小卷积核尺寸是 CNN 发展的一种趋势，这一定程度说明小卷积核、大深度的设计是有意义的。这一点很重要，我们将在附录中尝试解释它的原因。

10.4　GoogLeNet

"We Need To Go Deeper."

——*Inception*，2010

Christian Szegedy 等人在论文 *Going Deeper with Convolutions*[4]中描述了 GoogLeNet，并凭之夺得 ILSVRC-2014 冠军。GoogLeNet 使用了一种称为 Inception 的模块，该模块得名于 2010 年的电影 *Inception*（盗梦空间）。Inception 模块的结构如图 10-9 所示。

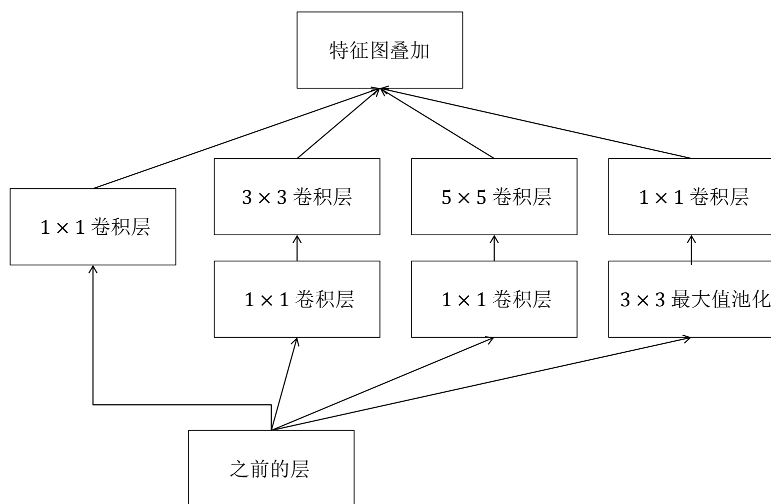


图 10-9　Inception 模块

Inception 模块将输入发送给 4 个部分，这 4 个部分从左至右依次如下：

- 卷积核为 1 × 1 的卷积层；
- 卷积核为 1 × 1 卷积层，后连卷积核为 3 × 3 卷积层；
- 卷积核为 1 × 1 卷积层，后连卷积核为 5 × 5 卷积层；
- 3 × 3最大值池化层，后连卷积核为 1 × 1 卷积层。

所有卷积层和池化层都采用补零填充且步幅为 1，它们都不改变图像的尺寸。各个卷积层的卷积核数量可任意设置。最后，Inception 模块将 4 个部分输出的特征图叠在一起作为输出。可以将 Inception 模块视为一个大卷积层，这个大卷积层的卷积核尺寸不止一种。通常的卷积层不论有多少卷积核，它们的尺寸都是相同的，而 Inception 模块相当于允许一个卷积层内包含不同尺寸的卷积核，以提取不同尺度的特征。

GoogLeNet 将卷积层、池化层等组件与 Inception 模块连接，组成深度 CNN 结构，它的结构如表 10-5 所示。

表 10-5　GoogLeNet 的结构

类型（名称）	尺寸/步幅	输出尺寸	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	#1×1 pooling
输入	—	224 × 224 × 3	—	—	—	—	—	—
卷积	7 × 7/2	112 × 112 × 64	—	—	—	—	—	—
最大值池化	3 × 3/2	56 × 56 × 64	—	—	—	—	—	—
卷积	1 × 1/1	56 × 56 × 64						
卷积	3 × 3/1	56 × 56 × 192	—	—	—	—	—	—
最大值池化	3 × 3/2	28 × 28 × 192	—	—	—	—	—	—
Inception	—	28 × 28 × 256	64	96	128	16	32	32
Inception	—	28 × 28 × 480	128	128	192	32	96	64
最大值池化	3 × 3/2	14 × 14 × 480	—	—	—	—	—	—
Inception	—	14 × 14 × 512	192	96	208	16	48	64
Inception	—	14 × 14 × 512	160	112	224	24	64	64
Inception	—	14 × 14 × 512	128	128	256	24	64	64
Inception	—	14 × 14 × 528	112	144	288	32	64	64
Inception	—	14 × 14 × 832	256	160	320	32	128	128
最大值池化	3 × 3/2	7 × 7 × 832	—	—	—	—	—	—
Inception	—	7 × 7 × 832	256	160	320	32	128	128
Inception	—	7 × 7 × 1024	384	192	384	48	128	128
平均值池化	7 × 7/1	1 × 1 × 1024	—	—	—	—	—	—
全连接	—	1 × 1 × 1000	—	—	—	—	—	—
SoftMax	—	1 × 1 × 1000	—	—	—	—	—	—

GoogLeNet 的卷积层和池化层采用补零填充，卷积层的激活函数是 ReLU。表 10-5 第一列是层的类型，第二列是卷积层或池化层的尺寸和步幅，第三列是输出尺寸和通道数，最后 6 列描述 Inception 模块各部分的卷积核数量，其中各列对应的部分如下：

- #1 × 1，1 × 1 卷积层；
- #3 × 3 reduce，3 × 3 卷积层之前的 1 × 1 卷积层；
- #3 × 3，3 × 3 卷积层；
- #5 × 5 reduce，5 × 5 卷积层之前的 1 × 1 卷积层；
- #5 × 5，5 × 5 卷积层；

- ❑ #1 × 1 pooling，池化层之后的 1 × 1 卷积层。

将这 6 列的第一、三、五、六列相加，就是该 Inception 模块的输出特征图数。GoogLeNet 对第一个最大值池化层和第三个卷积层施加局部响应标准化，对全连接层施加 Dropout，Dropout 率 0.4。

训练时，在第三和第六个 Inception 模块后连接了两个小规模的辅助 CNN，它们包括一个 5 × 5、步幅 3 的平均值池化层，后接 1 × 1、步幅 1 的卷积层，再连两个全连接层，最后施加 SoftMax 函数。卷积层和池化层采用补零填充，卷积层和第一个全连接层的激活函数是 ReLU。将辅助 CNN 的交叉熵乘以系数 0.3 后加到总损失上。辅助 CNN 的目的是缓解梯度消失，使前面的层容易得到训练。GoogLeNet 的结构如图 10-10 所示。



图 10-10 GoogLeNet 示意图[4]

图 10-10　（续）

10.5 ResNet

Kaiming He, Xiangyu Zhang, Shaoqing Ren 和 Jian Sun 在论文 *Deep Residual Learning for Image Recognition*[5]中提出了残差学习，并设计了几种统称为 ResNet 的 CNN。利用跳跃连接，将若干层的输入与它们的输出逐元素相加作为最终的输出，这就是一个残差单元（residual unit，RU）。作者使用的两种残差单元如图 10-11 所示。

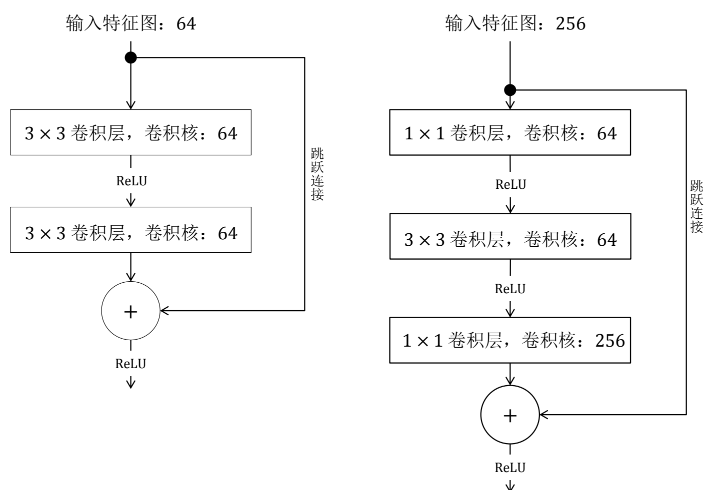


图 10-11 残差单元

图 10-11 左侧的残差单元包含两个卷积层，它们的卷积核尺寸是 3 × 3，卷积核数量是 64，激活函数为 ReLU。它们采用补零填充且步幅为 1，不改变图像尺寸。它们的卷积核数量与输入通道数相同，都是 64。这两个卷积层的输出与输入形状相同，可以逐元素相加。右侧的残差单元包含三个卷积层，1 × 1 卷积层压缩通道数，之后是一个 3 × 3 卷积层，最后的 1 × 1 卷积层有 256 个卷积核，与输入通道数相同。34 层的 ResNet-34 如图 10-12 所示。



图 10-12 ResNet-34 结构示意图[5]

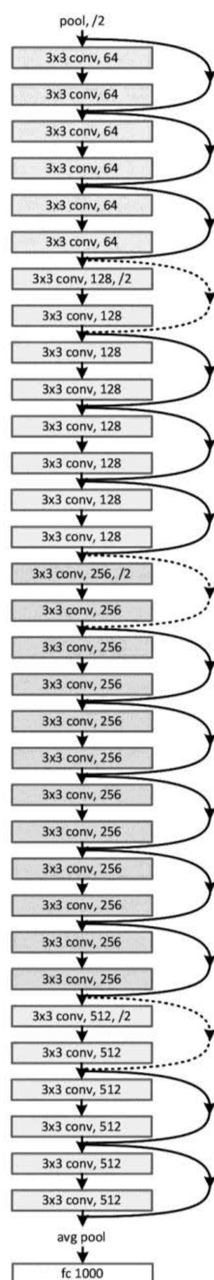


图 10-12　（续）

ResNet-34 的输入图像连接到一个包含 64 个 7 × 7 卷积核的卷积层，步幅为 2，后连接步幅为 2 的最大值池化层，将图像尺寸压缩一半，之后是 16 个残差单元。虚线标识的跳跃连接包含

1×1卷积层以调整数据形状。残差单元逐渐减小图像尺寸，增加通道数，最后经过步幅为1的平均值池化层后连接到1000个神经元的全连接层，施加SoftMax得到1000类的概率（图中没有显示）。表10-6展示5种不同深度的ResNet。

表10-6 5种不同深度的ResNet

模　块	输出尺寸	18层ResNet	34层ResNet	50层ResNet	101层ResNet	152层ResNet
卷积层	112×112	7×7卷积核，64通道，步幅2				
最大值池化	56×56	3×3，步幅2				
残差单元	56×56	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$
残差单元	28×28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times8$
残差单元	14×14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times36$
残差单元	7×7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$
平均值池化	7×7	7×7，步幅1				
全连接	1000	—				
SoftMax	1000	—				

10.6 小结

本章按时间顺序依次介绍了5种CNN，这些CNN在ILSVRC竞赛中的表现越来越好。竞赛是一种选择压力，在这种压力下，CNN不断变异、适者生存。我们能看出一个大致趋势：CNN的深度加深而卷积核变小。这表明深度确实是一个至关重要的因素，深度给CNN带来的能力提升，不是仅仅用高自由度加大样本量就能解释的，这也是脱胎于传统机器学习的深度学习能够自成一门子学科的原因。

不过目前说深度学习是一门学科还有些站不住脚，因为我们还没有可以解释深度本质的理论框架，这也是深度学习被诟病为炼金术的原因。但即便是炼金术，也是理论指导下的实践，而不是盲目尝试。无论是四元素理论，还是汞–硫理论，哪怕后来被证明是错误的，人们也从没放弃对理论的追求。出于这个动机，本书附录将从元胞自动机和动力学角度尝试分析一下“深度”的意义，抛砖引玉，希望对读者有所启发。

第 11 章

TensorFlow 实例

本章从一个实际问题入手，介绍如何使用 TensorFlow 框架构建和训练模型。本章并非 TensorFlow 的使用指南，也不是模型调优的实践指导，我们旨在以 TensorFlow 为工具，将本书介绍的原理和概念具象化，加深读者的理解。

MNIST 图像数据集包含阿拉伯数字 0~9 的手写图像，样本是 28×28 的单通道灰度图像，像素取值$[0,1]$区间内的实数，表示灰度。若将样本像素阵列的行连接起来，可形成一个 784 维特征向量，用$\boldsymbol{x}_{784}$表示。样本的类别是 $0,1,2,\cdots,9$ 十个数字。用向量$\boldsymbol{y}$表示类别的 One-Hot 编码。例如，若类别是 3，则向量$\boldsymbol{y}$是$(0,0,0,1,0,0,0,0,0,0)^{\mathrm{T}}$，$\boldsymbol{y}$的第 4 分量（以 0 开始，索引为 3 的分量）为 1，其余分量为 0。

本章包括的模型有多分类逻辑回归、多层全连接神经网络、LeNet-5、AlexNet 和 VGG16。我们用 Python 代码展示这些模型的构建、训练和评估。正文包括核心代码段落以及讲解，完整的代码见代码库：gitee.com/neural_network/neural_network_code。

11.1 多分类逻辑回归

第 1 章介绍的逻辑回归模型解决的是二分类问题，为解决多个类别的分类问题，我们可以将多个二分类逻辑回归组合起来，形成复合模型。最简单的一种组合办法是 OvO（one vs others）：如果问题有K个类别，则训练K个二分类逻辑回归模型，其中第k个模型区分第k类与其他类，预测时将样本提交给这K个模型，取输出概率最大的模型所识别的类别作为预测类别。

OvO 虽然简单，但却存在一些问题。首先，各个模型也许能很好地区分某一类与其他类，但它们的输出概率之间不具可比性。其次，如果每个类别的样本数量相当，那么每一个模型都将面对正负样本不均衡的问题。最后，当样本数和类别数很大时，训练多个模型的计算量也是不小的负担。

本节采用另一种方式——多分类逻辑回归（multinomial logistic regression，MLR）。我们首先回忆二分类逻辑回归，它假设正负类的对数概率比是关于输入向量的仿射函数：

$$\log\frac{p_{\mathrm{p}}}{p_{\mathrm{n}}}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b \tag{11.1}$$

MLR 遵循同样的思路，它认为K个类别的概率的对数是输入向量$\boldsymbol{x}$的仿射函数：

$$\log p_k=\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}+b_k-\log Z\ ,\quad k=1,2,\cdots,K \tag{11.2}$$

其中，$-\log Z$是归一化因子。K个权值向量$\boldsymbol{w}_k$和偏置b_k可取任意值，但必须保证p_k（$k=1,2,\cdots,K$）是合法的概率分布，即$\sum_{k=1}^{K}p_k=1$，这就需要归一化因子。至于Z的值是什么，为什么采用$-\log^Z$的形式，暂时按下不表。我们先看一看任意两个类别的对数概率比：

$$\log\frac{p_i}{p_j}=\log p_i-\log p_j=\left(\boldsymbol{w}_i-\boldsymbol{w}_j\right)^{\mathrm{T}}\boldsymbol{x}+\left(b_i-b_j\right) \tag{11.3}$$

可以看到，和二分类逻辑回归一样，两个类别的对数概率比是关于输入向量$\boldsymbol{x}$的仿射函数，其法向量是$\boldsymbol{w}_i-\boldsymbol{w}_j$，截距是$b_i-b_j$，所以 MLR 模型对任意两个类别的分界面是超平面，如图 11-1 所示。

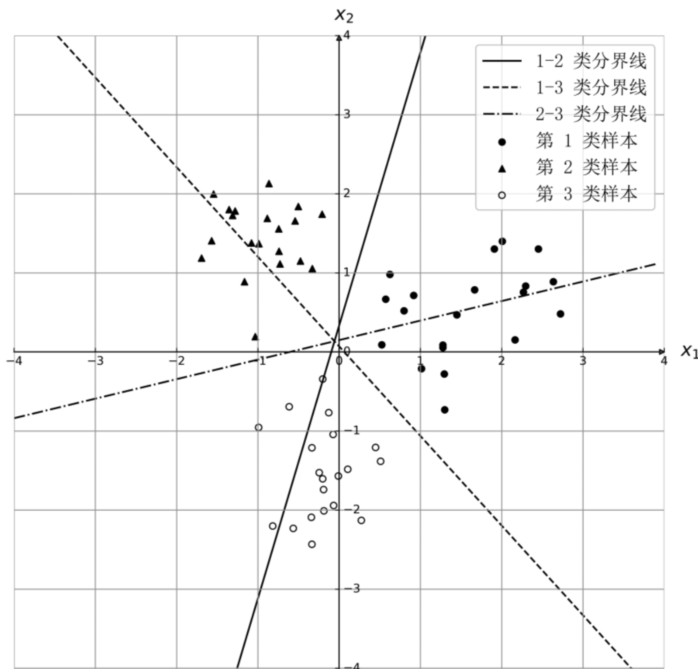


图 11-1 MLR 的分界线

现在我们回过头来考察一下Z的取值，将式（11.2）变形，可得到：

$$p_k = \frac{1}{Z} \cdot \mathrm{e}^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}+b_k}, \quad k = 1,2,\cdots,K \tag{11.4}$$

由于必须满足$\sum_{k=1}^{K} p_k = 1$，所以必然有：

$$Z = \sum_{k=1}^{K} \mathrm{e}^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}+b_k} \tag{11.5}$$

将式（11.5）代回式（11.4），得到：

$$p_k = \frac{\mathrm{e}^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}+b_k}}{\sum_{i=1}^{K} \mathrm{e}^{\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}+b_i}}, \quad k = 1,2,\cdots,K \tag{11.6}$$

这是对K个仿射函数值$\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x} + b_k$施加 SoftMax 函数。如此看来，MLR 其实是对输出施加 SoftMax 函数的单层神经网络，如图 11-2 所示。

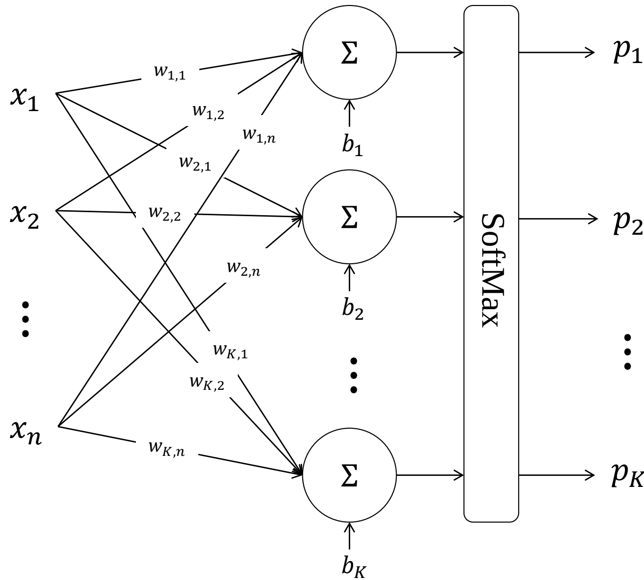


图 11-2　MLR 是单层神经网络

若输入向量为n维，以K个n维权值向量$\boldsymbol{w}_k$为行，构造$K \times n$的权值矩阵$\boldsymbol{W}$，以K个偏置为分量组成偏置向量$\boldsymbol{b}$，则 MLR 的计算可表达为：

$$p(\boldsymbol{x}) = \mathrm{SoftMax}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{11.7}$$

向量$p(\boldsymbol{x})$的各个分量是 MLR 对样本向量$\boldsymbol{x}$计算的各类别概率，取最大的概率对应的类别为

MLR 对样本的预测类别。相比于 OvO，MLP 的输出是合法的多分类概率分布。训练 MLR 时采用交叉熵作为损失函数，计算样本的类别 One-Hot 编码与$p(\boldsymbol{x})$之间的交叉熵：

$$\text{Loss}(\boldsymbol{W},\boldsymbol{b}|\boldsymbol{x},\boldsymbol{y}) = -\sum_{i=1}^{K} y_i \log p(\boldsymbol{x})_i \quad (11.8)$$

我们对权值（而不对偏置）施加$\mathcal{L}_2$正则化，带$\mathcal{L}_2$正则项的损失函数是：

$$\text{Loss}(\boldsymbol{W},\boldsymbol{b}|\boldsymbol{x},\boldsymbol{y}) = -\sum_{i=1}^{K} y_i \log p(\boldsymbol{x})_i + \lambda \sum_{i=1}^{K} \sum_{j=1}^{n} w_{i,j}^2 \quad (11.9)$$

现在把损失函数加入计算图，得到从输入向量到损失值的整个计算图，如图 11-3 所示。

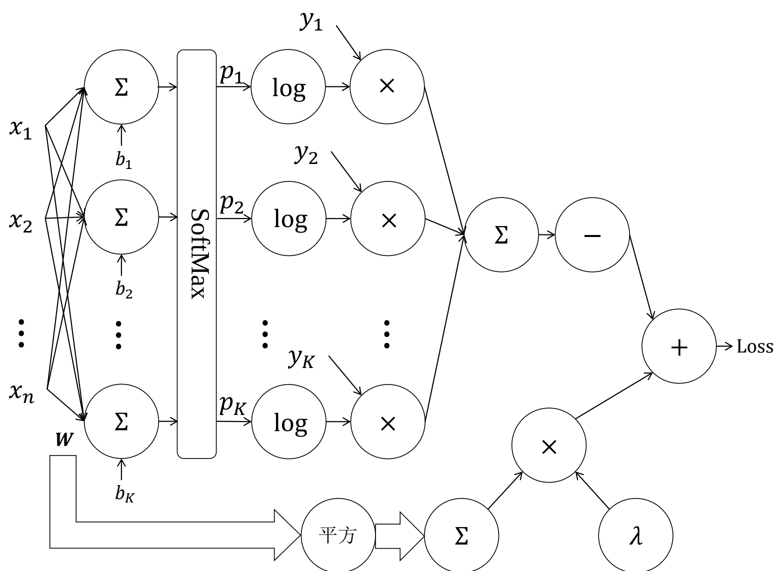


图 11-3 带损失值的 MLR 计算图

我们现在进入编码环节，首先，我们展示如何使用 TensorFlow 构造 MLR（不带损失值）的计算图，代码如下所示：

```
# 构造 MLR 的计算图
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
W = tf.Variable(tf.truncated_normal([784, 10], stddev=0.01))
b = tf.Variable(tf.zeros([10]))
p = tf.nn.softmax(tf.matmul(X, W) + b)
```

TensorFlow 中的张量（tensor）可以是标量、向量、矩阵以及更高维度的阵列。张量是 TensorFlow 计算图的节点。计算图的前向传播过程好似 Tensor 向前流动，这就是 TensorFlow 的含义。MLR 的计算图有 5 个张量：X、y、W、b 和 p。

X 是占位符（placeholder），训练或预测时将样本向量赋值给 X。X 有 2 个维度，第二维度是样本向量的维数——784，第一维度是 None，表示未定，允许将不定数量的样本赋给它。训练或预测时，可以将一批多个样本提交给计算图。占位符 y 的第一维度也未定，第二维度是 10，是类别 One-Hot 编码向量的维数。训练时将一批样本的类别 One-Hot 编码向量赋给 y，用于计算交叉熵。

变量 W 是 784×10 的权值矩阵。W 的元素用 tf.truncated_normal 函数初始化，它以 0 均值、0.01 标准差的正态分布初始化矩阵的每个元素，若随机值的绝对值超过两倍标准差，则被截断。变量 b 是 10 维偏置向量，以 0 值初始化。

用矩阵乘法 tf.matmul 计算 XW，这是 ?×784 的矩阵乘以 784×10 的矩阵，得到 ?×10 的矩阵。加上偏置向量 b 后，得到仿射函数值向量。注意+运算符经过重载，执行的是张量之间的加法，而且自动广播（broadcast），给 XW 的每一行加上 10 维偏置向量。最后，对 XW+b 施加 tf.nn.softmax 得到张量 p。p 是 ?×10 的矩阵，包含模型对一批输入样本的预测概率。

注意，上述代码段是在构造计算图，并不是真正执行计算：可以在一个会话中对计算图进行计算，即求某个张量的值，前提是被求值的张量的上游张量都已经有值——占位符已被赋值，变量已被初始化。

以下代码计算 MLR 对 MNIST 测试集样本的预测概率：

```
# 前向计算 MLR
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("mnist_dataset/", one_hot = True)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    prob = sess.run(p, feed_dict={X: mnist.train.images})
```

首先创建一个会话并初始化变量，接着求张量 p 的值，这时需要将 MNIST 测试集样本赋给占位符 X。若测试集包含n个样本，则赋给 X 的是 $n\times 784$ 矩阵。

这时得到的概率是无用的，因为模型的权值和偏置是随机的。若要训练 MLR，首先需要扩展计算图，将损失值加入其中。首先计算 p 的各分量的对数，之后计算类别 One-Hot 编码与对数

概率向量的内积，最后将结果取负，这就是交叉熵。我们还需要将$\mathcal{L}_2$正则项加到损失上，计算权值矩阵 W 所有元素的平方和，乘以正则化强度。构造损失值计算图的代码如下所示：

```
# 构造带正则项的损失值

l2_lambda = 0.0005
cost_cross_entropy = tf.reduce_mean(-tf.reduce_sum(y * tf.log(tf.clip_by_value(p, 1e-10, 1.0)), axis=1))
cost_regularization_l2 = l2_lambda * tf.reduce_sum(tf.square(W))
loss = tf.add(cost_cross_entropy, cost_regularization_l2)
```

tf.clip_by_value 将 p 的元素截断在[1e-10, 1.0]区间，这么做是为了防止概率超过 1 或过小（导致计算不稳定）。tf.reduce_sum 可以沿任一维度计算张量元素的和，若没有指定 axis 参数，则计算所有元素的和。张量 p 和占位符 y 的形状都是 ?×10，它们分别包含一批样本的预测概率和类别 One-Hot 编码。首先计算 p 的每个元素的对数，再与 y 逐元素相乘，沿着第 2 维度相加并取反，得到的 ?×1 张量是这批样本的交叉熵。调用 tf.reduce_mean 求这批样本的平均交叉熵。tf.square 计算张量 W 每一个元素的平方，tf.reduce_sum 求它们的和，乘上正则化强度后得到正则项，正则项与交叉熵相加就是损失值。

至此我们已经构建了从输入到损失值的完整计算图。训练，就是将损失值作为权值和偏置的函数，用梯度下降法更新权值和偏置。TensorFlow 提供了对计算图进行前向和反向传播、计算梯度，并利用梯度更新变量值的优化器，如下代码所示：

```
# 构造梯度下降优化器

learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
one_gd_step = optimizer.minimize(loss)
```

用 tf.train.GradientDescentOptimizer 构造一个梯度下降优化器，学习率是 0.01。以损失值张量 loss 为参数调用该优化器的 minimize 方法，得到一个优化运算 one_gd_step。在会话中每执行一次 one_gd_step，框架用一批样本前向传播计算损失值，反向传播计算梯度，并用梯度更新所有变量，这就是 Mini Batch 梯度下降的一次迭代。训练过程的代码如下：

```
# 运用梯度下降法训练 MLR

batch_size = 64
epoch = 50

with tf.Session() as sess:
for i in range(epoch):
```

```
    number_batch = int(mnist.train.num_examples / batch_size)
    for j in range(number_batch):
        train_x, train_y = mnist.train.next_batch(batch_size)
        _ = sess.run(one_gd_step, feed_dict={X: train_x, y: train_y})
```

用训练集样本总数和 batch_size 计算每一个 epoch 的迭代次数，调用 mnist.train.next_batch 获得下一批样本和类别 One-Hot 编码，将它们赋给占位符。在会话中执行一次 one_gd_step，完成一步梯度下降。每一步梯度下降后，可以计算当前模型在训练集和测试集上的正确率，代码如下所示：

```
# 计算当前模型在训练集和测试集上的正确率

correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    train_acc = sess.run(accuracy, feed_dict={X: mnist.train.images, y: mnist.train.labels})
    test_acc = sess.run(accuracy, feed_dict={X: mnist.test.images, y: mnist.test.labels})
```

上述代码取预测概率和类别 One-Hot 编码的最大值索引，它们分别对应着预测类别和真实类别。调用 tf.equal 判断它们是否相等，将布尔值转换为浮点值后求平均，就得到了一批样本上的正确率。在会话中，分别将训练集和测试集的样本与类别 One-Hot 编码赋给占位符，得到训练集和测试集上的正确率。另外，还可以计算模型在训练集和测试集上的损失值，代码如下所示：

```
# 计算当前模型在训练集和测试集上的损失值

with tf.Session() as sess:
    train_loss = sess.run(loss, feed_dict={X: mnist.train.images, y: mnist.train.labels})
    test_loss = sess.run(loss, feed_dict={X: mnist.test.images, y: mnist.test.labels})
```

训练过程中，每一步迭代后都可以计算当前模型在训练集和测试集上的损失值与正确率。训练结束后，可将训练集和测试集上的损失值和正确率随迭代变化的曲线绘制出来，如图 11-4 所示。

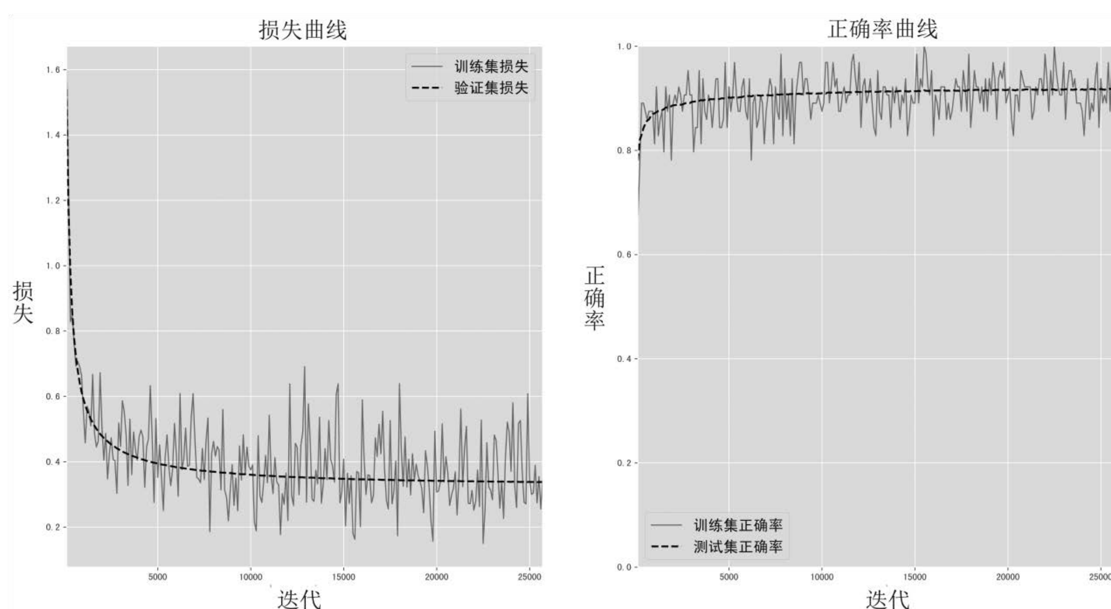


图 11-4　MLR 的损失值曲线、正确率曲线

因为训练集太大，所以每一步迭代后是在当前 Mini Batch 上计算正确率和损失值，故曲线较毛糙。测试集的正确率和损失值是在全部测试样本上计算的，曲线较平滑。训练完成的 MLR 在测试集上的混淆矩阵如图 11-5 所示。

混淆矩阵

真实 \ 预测	0	1	2	3	4	5	6	7	8	9
0	961	0	2	2	0	4	8	1	2	0
1	0	1105	2	2	1	2	4	2	17	0
2	10	9	904	18	14	1	14	14	40	8
3	3	1	19	922	0	24	3	11	16	11
4	1	3	4	1	915	0	11	2	8	37
5	10	4	2	41	9	766	17	6	28	9
6	12	3	3	2	11	13	911	1	2	0
7	3	13	22	7	8	0	0	939	3	33
8	7	10	8	27	9	25	13	13	849	13
9	12	8	3	12	36	7	0	22	6	903

图 11-5　MLR 在测试集上的混淆矩阵

MLR 的正确率是 91.8%，这并不是一个足够好的正确率。可以看到，测试集上的损失值曲线没有上扬，说明没有过拟合，模型仍处于欠拟合状态。对于 MNIST 数据集来说，MLR 的自由度是不够的。

11.2　多层全连接神经网络

MLR 相当于单层全连接神经网络，我们现在为 MLR 添加一个隐藏层，形成多层全连接神经网络 MLP，如图 11-6 所示。

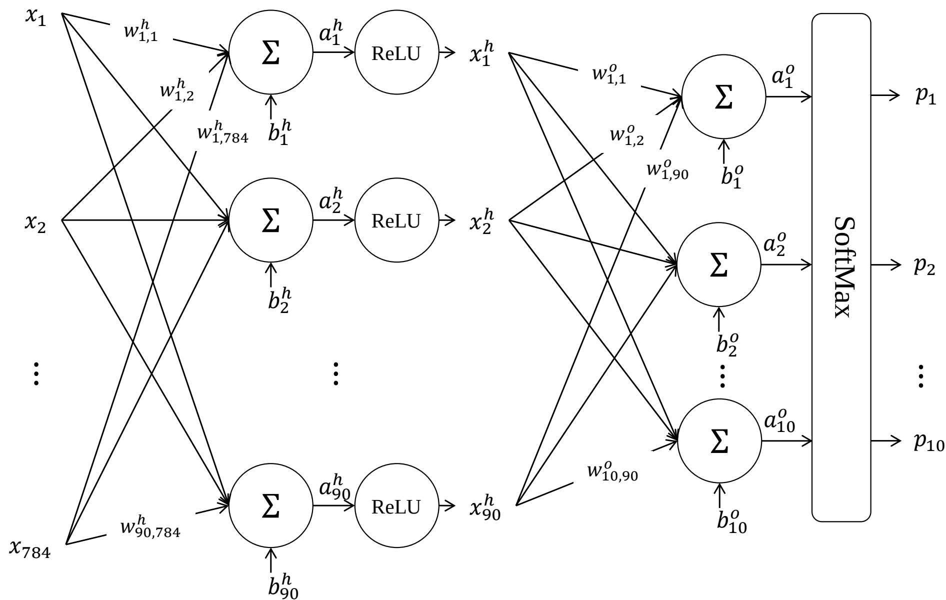


图 11-6　单隐藏层的全连接神经网络

这个 MLP 以 784 维向量为输入，隐藏层神经元的数量取 90，这是占样本总方差 90%的主成分数量。既然主成分反映了样本的真实维数，我们期待 MLP 的隐藏层能够学习并利用这一点。

令$\boldsymbol{W}^h$是 90×784 的矩阵，每一行是隐藏层神经元的权值向量，$\boldsymbol{b}^h$是 90 维向量，每个分量是隐藏层神经元的偏置。令$\boldsymbol{W}^o$是 10×90 的矩阵，每一行是输出层神经元的权值向量，$\boldsymbol{b}^o$是 10 维向量，每个分量是输出层神经元的偏置。用矩阵形式表达该 MLP 就是：

$$p(\boldsymbol{x}_{784}) = \text{SoftMax}(\boldsymbol{W}^o \cdot \text{Logistic}(\boldsymbol{W}^h \boldsymbol{x}_{784} + \boldsymbol{b}^h) + \boldsymbol{b}^o) \tag{11.10}$$

我们构造这个 MLP 的计算图，代码如下：

```
# 构造 MLP 计算图

X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

Wh = tf.Variable(tf.truncated_normal([784, 90], stddev=0.01))
bh = tf.Variable(tf.zeros([90]))
h1 = tf.nn.relu(tf.matmul(X, Wh) + bh)

Wo = tf.Variable(tf.truncated_normal([90, 10], stddev=0.01))
bo = tf.Variable(tf.zeros([10]))

p = tf.nn.softmax(tf.matmul(h1, Wo) + bo)
```

权值以 0 均值、0.01 标准差的正态分布初始化，绝对值超过两倍标准差则截断。偏置初始化为 0 值。`tf.nn.relu` 是 ReLU 激活函数。交叉熵的构造与 MLR 相同。我们对 MLP 的两个权值矩阵施加$\mathcal{L}_2$正则化。$\mathcal{L}_2$正则项是两个权值矩阵所有元素的平方和，再乘以正则化强度。带$\mathcal{L}_2$正则项的损失函数是：

$$\text{Loss}(\boldsymbol{W}^{h,o},\boldsymbol{b}^{h,o}|\boldsymbol{x}_{784},\boldsymbol{y}) = -\sum_{i=1}^{K} y_i \log^{p(x_{784})_i} + \lambda\left(\sum\left(w_{i,j}^{h}\right)^2 + \sum\left(w_{i,j}^{o}\right)^2\right) \quad (11.11)$$

构造损失值张量的代码如下：

```
# 构造 MLP 带正则项的损失函数

l2_lambda = 0.0005
cost_cross_entropy = tf.reduce_mean(-tf.reduce_sum(y * tf.log(tf.clip_by_value(p, 1e-10, 1.0)), axis=1))
cost_regularization_l2 = l2_lambda * (tf.reduce_sum(tf.square(Wh)) + tf.reduce_sum(tf.square(Wo)))
loss = tf.add(cost_cross_entropy, cost_regularization_l2)
```

我们仍然使用 Mini Batch 梯度下降训练 MLP，训练部分的代码与 MLR 完全一致，这里不再重复。MLP 在训练集和测试集上的损失值和正确率随着迭代变化的曲线如图 11-7 所示。

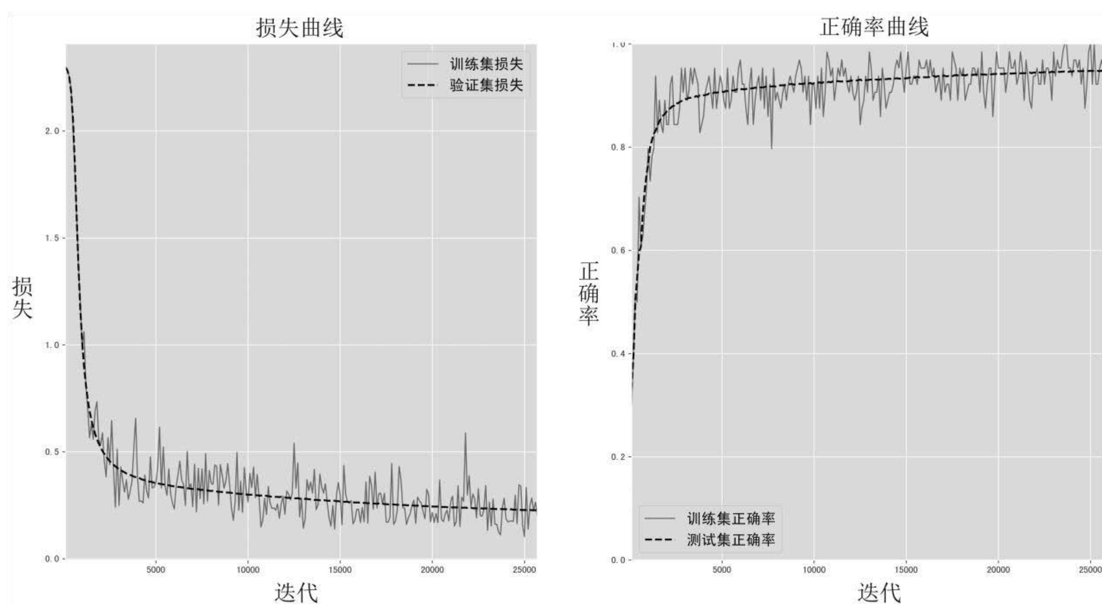


图 11-7　MLP 的损失值曲线、正确率曲线

训练完成的 MLP 在测试集上的混淆矩阵如图 11-8 所示。

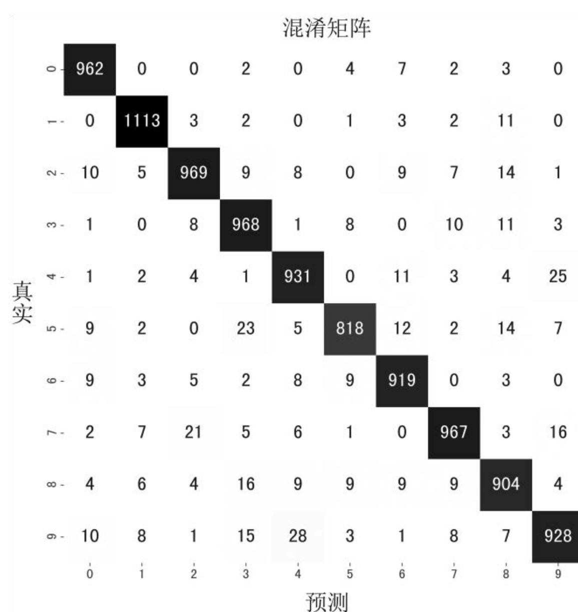


图 11-8　MLP 在测试集上的混淆矩阵

MLP 的正确率达到 95.1%，超过 MLR。如果要继续加大模型的自由度，可以加大 MLP 的深度。两隐藏层 MLP 执行的计算是：

$$p(\boldsymbol{x}_{784}) = \text{SoftMax}(\boldsymbol{W}^3 \cdot \text{Logistic}(\boldsymbol{W}^2 \cdot \text{Logistic}(\boldsymbol{W}^1\boldsymbol{x}_{784} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) \quad (11.12)$$

可根据式（11.12）构造计算图，并将新增的权值矩阵的正则项加到损失上。

11.3 LeNet-5

我们构造一个修改版 LeNet-5，它与第 10 章描述的 LeNet-5 有几点不同。

- 卷积层和全连接层采用 ReLU，而不是 Tanh；
- 池化层不带激活函数；
- 输出层采用 SoftMax，而不是高斯径向基；
- S3 和 C3 之间全连接；
- 以 28 × 28 原始图像为输入，C1 采用补零填充，而不是舍弃填充。

修改版 LeNet-5 的结构如表 11-1 所示。

表 11-1 修改版 LeNet-5 的结构

层	类 型	核/池化尺寸	步 幅	填 充	输出尺寸	输出通道数	激活函数
输入层	—	—	—	—	28 × 28	1	—
C1	卷积	5 × 5	1	补零	28 × 28	6	ReLU
S2	平均值池化	2 × 2	2	补零	14 × 14	6	—
C3	卷积	5 × 5	1	舍弃	10 × 10	16	ReLU
S4	平均值池化	2 × 2	2	补零	5 × 5	16	—
C5	卷积	5 × 5	1	舍弃	1 × 1	120	ReLU
F6	全连接	—	—	—	84	—	ReLU
输出层	全连接	—	—	—	10	—	SoftMax

构建该 LeNet-5 的计算图的代码如下：

```
# 构造 LeNet-5 的计算图

X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
```

```
X = tf.reshape(X, [-1, 28, 28, 1])

# 第一层：卷积层输出 28*28*6
conv1_w = tf.get_variable('conv1_w', [5, 5, 1, 6],
initializer=tf.truncated_normal_initializer(stddev=0.05))
conv1_b = tf.get_variable('conv1_b', [6], initializer=tf.constant_initializer(0.0))
conv1 = tf.nn.conv2d(X, conv1_w, strides=[1, 1, 1, 1], padding='SAME')
relu1 = tf.nn.relu(tf.nn.bias_add(conv1, conv1_b))

# 第二层：平均值池化层输出 14*14*6
pool2 = tf.nn.avg_pool(relu1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第三层：卷积层输出 10*10*16
conv3_w = tf.get_variable('conv3_w', [5, 5, 6, 16], initializer=tf.truncated_normal_initializer
    (stddev=0.05))
conv3_b = tf.get_variable('conv3_b', [16], initializer=tf.constant_initializer(0.0))
conv3 = tf.nn.conv2d(pool2, conv3_w, strides=[1, 1, 1, 1], padding='VALID')
relu3 = tf.nn.relu(tf.nn.bias_add(conv3, conv3_b))

# 第四层：平均值池化层输出 5*5*16
pool4 = tf.nn.avg_pool(relu3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第五层：卷积层输出 1*1*120
conv5_w = tf.get_variable('conv5_w', [5, 5, 16, 120], initializer=tf.truncated_normal_initializer
    (stddev=0.05))
conv5_b = tf.get_variable('conv5_b', [120], initializer=tf.constant_initializer(0.0))
conv5 = tf.nn.conv2d(pool4, conv5_w, strides=[1, 1, 1, 1], padding='VALID')
relu5 = tf.nn.relu(tf.nn.bias_add(conv5, conv5_b))
reshaped_relu5 = tf.reshape(relu5, [-1, 120])

# 第六层：全连接层输出 84
fc6_w = tf.get_variable('fc6_w', [120, 84], initializer=tf.truncated_normal_initializer(stddev=0.05))
tf.add_to_collection('losses', tf.reduce_sum(tf.square(fc6_w)))
fc6_b = tf.get_variable('fc6_b', [84], initializer=tf.constant_initializer(0.0))
fc6 = tf.nn.relu(tf.matmul(reshaped_relu5, fc6_w) + fc6_b)

# 输出层输出 10
fco_w = tf.get_variable('fco_w', [84, 10], initializer=tf.truncated_normal_initializer(stddev=0.05))
tf.add_to_collection('losses',(tf.reduce_sum(tf.square(fco_w)))
fco_b = tf.get_variable('fco_b', [10], initializer=tf.constant_initializer(0.0))
fco = tf.matmul(fc6, fco_w) + fco_b

p = tf.nn.softmax(fco)
```

所有卷积核和全连接层的权值都以 0 均值、0.05 标准差的正态分布初始化，绝对值超过两倍标准差则截断，偏置初始化为 0 值。用 `tf.nn.conv2d` 对输入张量做卷积运算，之后加上偏置并施加 ReLU 激活函数，这就构造了卷积层。卷积核的形状是 4 维，例如[5, 5, 6, 16]。这 4 个维数的意思是：卷积核的尺寸是 5×5，卷积核的“厚度”，即输入通道数是 6，共有 16 个卷积核。16 个卷积核各有一个偏置，所以偏置向量的维数是 16。

`tf.nn.conv2d` 的参数 `strides` 包含 4 个数，分别是 batch 上的步幅、横向步幅、纵向步幅和输入通道上的步幅，一般 batch 和输入通道上的步幅固定为 1。根据 LeNet-5 的结构，卷积层的横向和纵向步幅是 1，所以 `strides` 参数是`[1, 1, 1, 1]`。C1 卷积层的填充方式是补零，C3 和 C5 卷积层的填充方式是舍弃。

`tf.nn.avg_pool` 是平均值池化，它也有 `strides` 参数。LeNet-5 池化层的步幅为 2，所以 `strides` 参数是`[1, 2, 2, 1]`。`ksize` 参数是池化窗口的尺寸，4 个数字对应 4 个维度，这里取`[1, 2, 2, 1]`。池化层的填充方式是补零。

全连接层的构造方法与 MLR 或 MLP 相同。注意，代码用 `tf.add_to_collection` 将两个全连接层权值矩阵的元素平方和添加到集合 `losses` 中。构造损失值时可用 `tf.get_collection` 取出这些项，乘上正则化强度后加到损失上。构造损失值张量的代码如下：

```
# 构造带正则项的损失函数

l2_lambda = 0.0008
cost_cross_entropy = tf.reduce_mean(-tf.reduce_sum(y * tf.log(tf.clip_by_value(p, 1e-10, 1.0)), axis=1))
loss = tf.add(cost_cross_entropy, l2_lambda * tf.add_n(tf.get_collection("losses")))
```

训练部分的代码与之前相同，这里不再重复。LeNet-5 在训练集和测试集上的损失值和正确率随着迭代变化的曲线如图 11-9 所示。

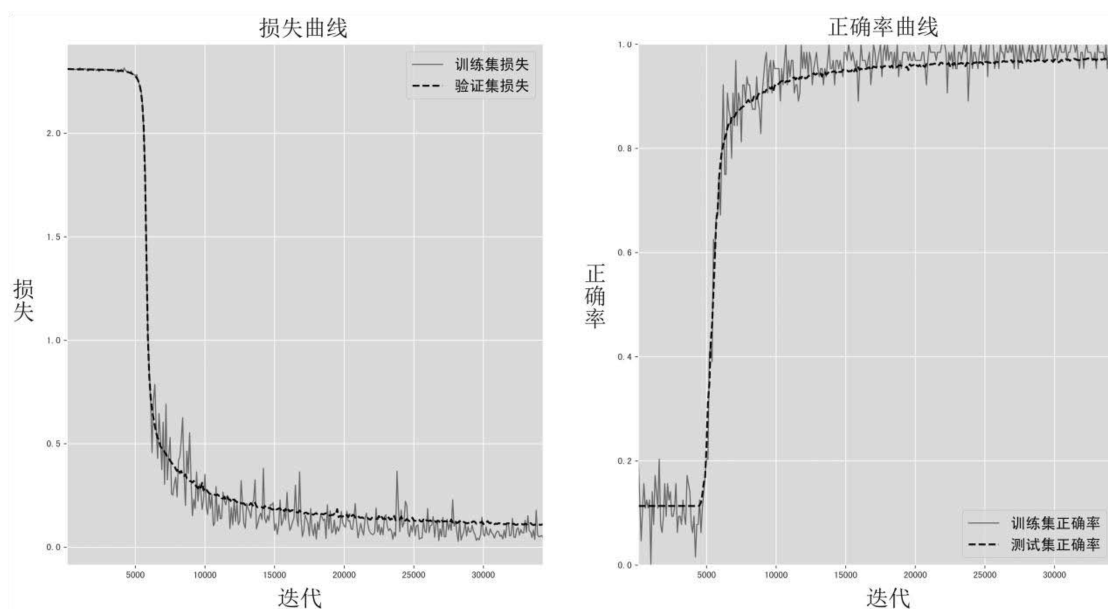
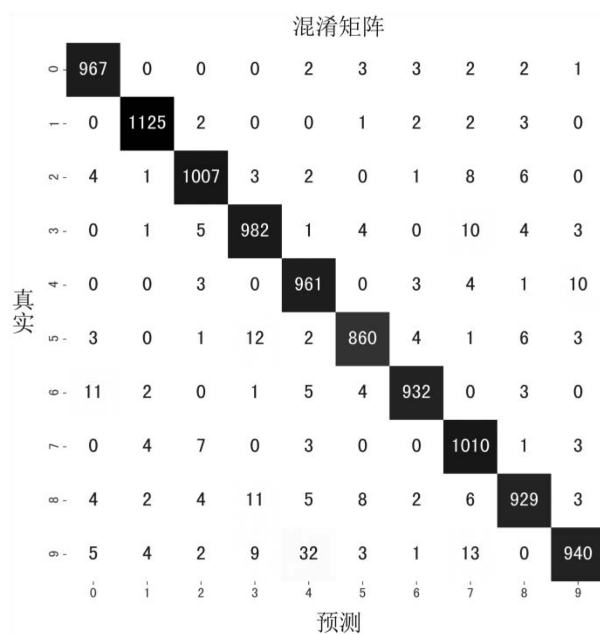


图 11-9　LeNet-5 的损失值曲线、正确率曲线

训练完成的 LeNet-5 在测试集上的混淆矩阵如图 11-10 所示。



真实 \ 预测	0	1	2	3	4	5	6	7	8	9
0	967	0	0	0	2	3	3	2	2	1
1	0	1125	2	0	0	1	2	2	3	0
2	4	1	1007	3	2	0	1	8	6	0
3	0	1	5	982	1	4	0	10	4	3
4	0	0	3	0	961	0	3	4	1	10
5	3	0	1	12	2	860	4	1	6	3
6	11	2	0	1	5	4	932	0	3	0
7	0	4	7	0	3	0	0	1010	1	3
8	4	2	4	11	5	8	2	6	929	3
9	5	4	2	9	32	3	1	13	0	940

图 11-10　LeNet-5 在测试集上的混淆矩阵

LeNet-5 的正确率达到了 97.8%，超过 MLR 和 MLP。我们没有精细地调优模型，这个正确率并非 LeNet-5 在 MNIST 上所能达到的最佳表现。

11.4 AlexNet

接下来我们尝试 AlexNet。相比于 LeNet-5，AlexNet 的复杂性上了一个台阶。AlexNet 的结构中重复出现卷积层、池化层以及全连接层，除了具体配置不同，构造每一层的计算图的方法是一致的，为此，我们首先定义几个辅助函数，代码如下：

```
# 辅助函数

def conv(input_tensor, name, kh, kw, dh, dw, n_output):
    """
    卷积操作

    参数
    ----------
    input_tensor: tf Tensor
    name: 卷积层名称
    kh: 卷积核高
    kw: 卷积核宽
    dh: 纵向步幅
    dw: 横向步幅
    n_output: 输出 size

    返回值
    ----------
    激活层
    """
    n_input = input_tensor.get_shape()[-1].value

    kernel = tf.get_variable(
        name=name + 'kernel',
        shape=[kh, kw, n_input, n_output],
        dtype=tf.float32,
        initializer=tf.truncated_normal_initializer(stddev=0.05))
    bias = tf.get_variable(
        name=name + 'bias', shape=[n_output], dtype=tf.float32, initializer=tf.constant_initializer
            (0.0))

    c = tf.nn.conv2d(input_tensor, kernel, (1, dh, dw, 1), padding='SAME')
    return tf.nn.relu(tf.nn.bias_add(c, bias), name=name)

def max_pool(input_tensor, name, kh, kw, dh, dw):
```

```
    """
    最大池化操作

    参数
    ----------
    input_tensor: tf Tensor
    name: 名称
    kh: 窗口高度
    kw: 窗口宽度
    dh: 纵向步幅
    dw: 横向步幅

    返回值
    ----------
    Tensor
    """

    return tf.nn.max_pool(input_tensor, ksize=[1, kh, kw, 1], strides=[1, dh, dw, 1], padding='SAME',
        name=name)

def fc(input_tensor, name, n_output):
    """
    全连接操作

    参数
    ----------
    input_tensor: tf Tensor
    name: 名称
    n_output: 输出 size

    返回值
    ----------
    激活层
    """

    n_input = input_tensor.get_shape()[-1].value
    weights = tf.get_variable(
        name=name + 'weights',
        shape=[n_input, n_output],
        dtype=tf.float32,
        initializer=tf.truncated_normal_initializer(stddev=0.05))
    tf.add_to_collection('losses', tf.nn.l2_loss(weights))
    bias = tf.get_variable(name=name + 'bias', shape=[n_output], dtype=tf.float32, initializer=
        tf.constant_initializer(0.0))

    return tf.nn.bias_add(tf.matmul(input_tensor, weights), bias)
```

这几个辅助函数以 `input_tensor` 张量为输入，构造卷积层、全连接层或最大值池化层。根据前文的讲解以及代码中的注释，读者应该能够理解这几个函数的功能，此处不再赘述。上述代码用 `tf.nn.l2_loss` 计算全连接层权值矩阵的元素平方和。构造 AlexNet 的计算图的代码如下：

```
# 构造 AlexNet 的计算图

X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

X = tf.reshape(X, [-1, 28, 28, 1])

# 卷积输入 shape 224*224*3 输出 shape 55*55*96
conv_1 = conv(X, 'conv_1', 11, 11, 4, 4, 96)

# 最大值池化输出 shape 27*27*96
pool_2 = max_pool(conv_1, 'pool_2', 3, 3, 2, 2)

# 卷积输出 shape 27*27*256
conv_3 = conv(pool_2, 'conv_3', 5, 5, 1, 1, 256)

# 最大值池化输出 shape 13*13*256
pool_4 = max_pool(conv_3, 'pool_4', 3, 3, 2, 2)

# 卷积输出 shape 13*13*384
conv_5 = conv(pool_4, 'conv_5', 3, 3, 1, 1, 384)
conv_6 = conv(conv_5, 'conv_6', 3, 3, 1, 1, 384)

# 卷积输出 shape 13*13*256
conv_7 = conv(conv_6, 'conv_7', 3, 3, 1, 1, 256)

# 最大值池化输出 shape 6*6*256
pool_8 = max_pool(conv_7, 'pool_8', 3, 3, 2, 2)

shape_size = 1
for i in range(1, 4):
    shape_size = shape_size * pool_8.get_shape()[-i].value
reshaped = tf.reshape(pool_8, [-1, shape_size], name='reshaped')

# 全连接层
fc_9 = fc(reshaped, 'fc_9', 4096)
relu_fc_9 = tf.nn.relu(fc_9)

fc_10 = fc(relu_fc_9, 'fc_10', 4096)
relu_fc_10 = tf.nn.relu(fc_10)
```

```
fc_output = fc(relu_fc_10, 'fc_output', 10)
p = tf.nn.softmax(fc_output)
```

这个计算图比 LeNet-5 复杂得多，但暴露在外的仍是输入样本占位符 X、类别 One-Hot 编码占位符 y 和预测概率向量 p，损失值张量和训练部分只关心这三个张量。构造损失值张量的代码和 LeNet-5 相同，不再重复。这次我们采用 Adam 优化器，代码如下：

```
# 构造 Adam 优化器

learning_rate = 0.005
optimizer = tf.train.AdamOptimizer(learning_rate)
one_gd_step = optimizer.minimize(loss)
```

用 `tf.train.AdamOptimizer` 构造 Adam 优化器，学习率为 0.005，其他超参数取默认值。训练部分的代码和之前一样，不再重复。AlexNet 在训练集和测试集上的损失值和正确率随着迭代变化的曲线如图 11-11 所示。

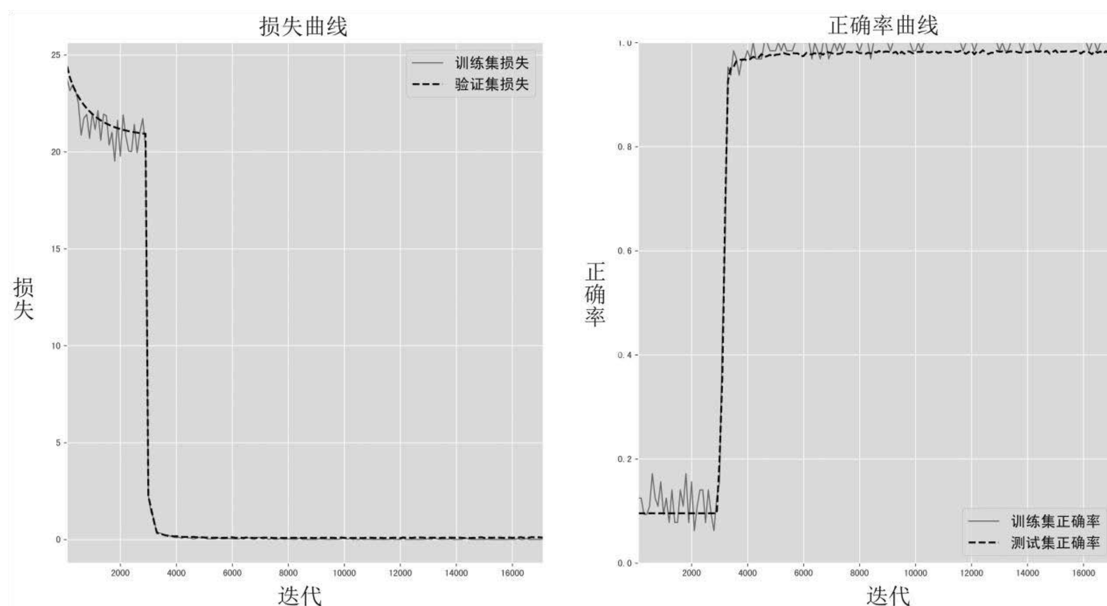


图 11-11　AlexNet 的损失值曲线、正确率曲线

训练完成的 AlexNet 在测试集上的混淆矩阵如图 11-12 所示。

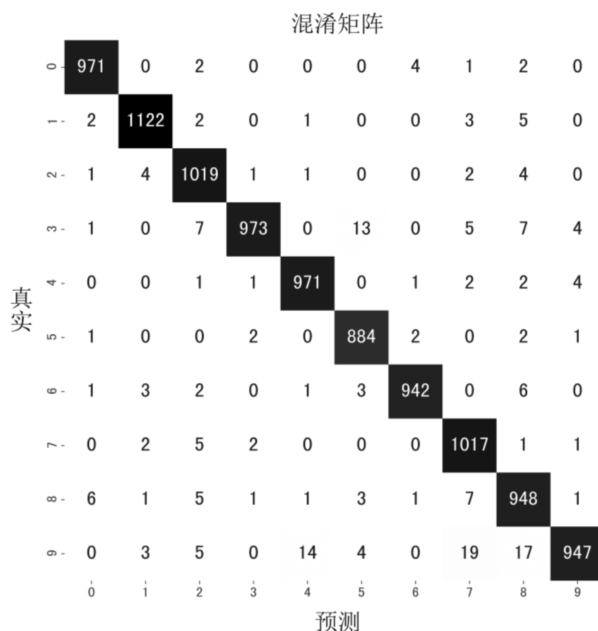


图 11-12　AlexNet 在测试集上的混淆矩阵

AlexNet 的正确率达到 98.8%，相对 LeNet-5 有提升。

11.5　VGG16

最后，我们尝试更深的网络：VGG16。本节构造的 VGG16 与第 10 章的描述有三点不同。

- 输入形状为 28 × 28 × 1；
- 输出层神经元个数为 10；
- 两个全连接层的神经元个数都为 128。

利用上一节的辅助函数构造 VGG16 的计算图，代码如下：

```
# 构造 VGG16 的计算图

X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

X = tf.reshape(X, [-1, 28, 28, 1])

# 卷积层组 1 输入 shape 224*224*3,输出 shape 224*224*64
conv_1_1 = conv(X, 'conv_1_1', 3, 3, 1, 1, 64)
```

```
conv_1_2 = conv(conv_1_1, 'conv_1_2', 3, 3, 1, 1, 64)

# 最大值池化操作输出 112*112*64
pool_1 = max_pool(conv_1_2, 'pool_1', 2, 2, 2, 2)

# 卷积层组 2 输出 112*112*128
conv_2_1 = conv(pool_1, 'conv_2_1', 3, 3, 1, 1, 128)
conv_2_2 = conv(conv_2_1, 'conv_2_2', 3, 3, 1, 1, 128)

# 最大值池化操作输出 56*56*128
pool_2 = max_pool(conv_2_2, 'pool_2', 2, 2, 2, 2)

# 卷积层组 3 输出 56*56*256
conv_3_1 = conv(pool_2, 'conv_3_1', 3, 3, 1, 1, 256)
conv_3_2 = conv(conv_3_1, 'conv_3_2', 3, 3, 1, 1, 256)
conv_3_3 = conv(conv_3_2, 'conv_3_3', 3, 3, 1, 1, 256)

# 最大值池化操作，输出 28*28*256
pool_3 = max_pool(conv_3_3, 'pool_3', 2, 2, 2, 2)

# 卷积层组 4 输出 28*28*512
conv_4_1 = conv(pool_3, 'conv_4_1', 3, 3, 1, 1, 512)
conv_4_2 = conv(conv_4_1, 'conv_4_2', 3, 3, 1, 1, 512)
conv_4_3 = conv(conv_4_2, 'conv_4_3', 3, 3, 1, 1, 512)

# 最大值池化操作输出 14*14*512
pool_4 = max_pool(conv_4_3, 'pool_4', 2, 2, 2, 2)

# 卷积层组 5 输出 14*14*512
conv_5_1 = conv(pool_4, 'conv_5_1', 3, 3, 1, 1, 512)
conv_5_2 = conv(conv_5_1, 'conv_5_2', 3, 3, 1, 1, 512)
conv_5_3 = conv(conv_5_2, 'conv_5_3', 3, 3, 1, 1, 512)

# 最大值池化操作输出 7*7*512
pool_5 = max_pool(conv_5_3, 'pool_5', 2, 2, 2, 2)

shape_size = 1
for i in range(1, 4):
    shape_size = shape_size * pool_5.get_shape()[-i].value
reshaped = tf.reshape(pool_5, [-1, shape_size], name='reshaped')

# 全连接层输出 10
fc_6 = fc(reshaped, 'fc_6', 128)
relu_fc_6 = tf.nn.relu(fc_6)
```

```
fc_7 = fc(relu_fc_6, 'fc_7', 128)
relu_fc_7 = tf.nn.relu(fc_7)

fc_8 = fc(relu_fc_7, 'fc_8', 10)

p = tf.nn.softmax(fc_8)
```

构造损失值张量和训练部分的代码与 AlexNet 相同，不再重复。VGG16 在训练集和测试集上的损失值和正确率随迭代变化的曲线如图 11-13 所示。

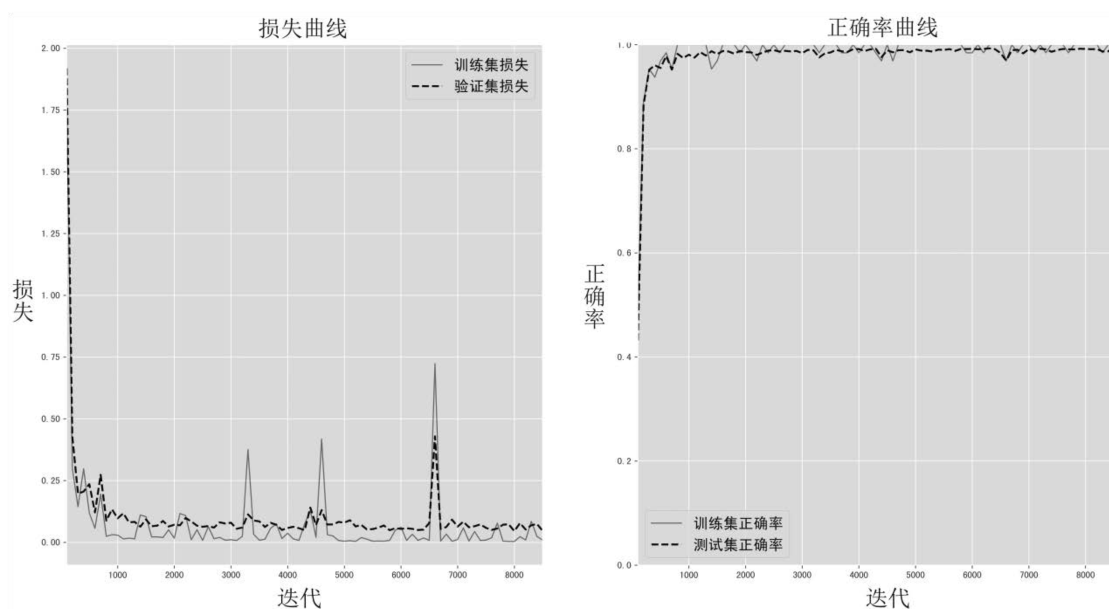


图 11-13　VGG16 的损失值曲线、正确率曲线

VGG16 在测试集上的正确率达到了 99.4%，已臻完美。训练完成的 VGG16 在测试集上的混淆矩阵如图 11-14 所示。

混淆矩阵

真实 \ 预测	0	1	2	3	4	5	6	7	8	9
0	974	0	0	0	0	1	0	1	4	0
1	0	1130	0	1	0	0	1	2	1	0
2	2	0	1021	0	0	0	1	4	4	0
3	0	0	1	1002	0	3	0	1	3	0
4	0	0	0	0	957	0	0	0	10	15
5	1	0	0	3	0	883	1	0	0	4
6	7	2	0	0	2	8	931	0	8	0
7	0	2	1	0	0	0	0	1018	1	6
8	0	0	0	0	0	0	0	0	974	0
9	0	0	0	0	1	1	0	1	21	985

图 11-14　VGG16 在测试集上的混淆矩阵

11.6　小结

本章围绕 MNIST 手写数字识别问题，展示了如何使用 TensorFlow 构造和训练多分类逻辑回归、多层全连接神经网络、LeNet-5、AlexNet 和 VGG16 模型。这 5 种模型从线性到非线性、从全连接到局部连接、从浅到深，表现也愈加优秀。MNIST 手写数字识别本质上是一个 784 维特征的 10 分类问题，作为图像识别问题，MNIST 相对简单，但作为传统机器学习问题，从特征维数和样本量来说，MNIST 属于较大型的问题。我们看到深度学习模型在 MNIST 上的表现明显超过传统模型。

本章的示例很简单，主要着重用代码展示相关的理论和概念，而并没有涉及专业级的 TensorFlow 编程和模型调优。无论是 TensorFlow 还是其他机器学习/深度学习框架，都是概念和原理的实现，只有深刻地掌握了原理，使用这些库时才能做到“恢恢乎其于游刃必有余地”，而不至于迷失在概念的迷宫中。“调包调库”并不简单，五花八门的模型和超参数就像是一部复杂机器的旋钮，只有理解原理才能知道每个旋钮的作用，从而精细地调节模型。

基础理论的演进没有表面的技术和工具的演进那么迅速，它们是相对稳定的，但也是最艰深的。为什么要掌握基础理论？在人工智能时代，新模型和新工具以井喷之势大量涌现，但底层的理论基础是共通的。只有深刻掌握基础理论，才能具备洞悉本质的眼光，迅速理解并精当使用这些新模型和新工具，这是一个机器学习工程师最大的本领和价值。

附录 A

CNN 与元胞自动机



第 10 章我们提到，在 CNN 的演化历程中存在卷积核变小而深度增加的趋势，局部性和深度是现代 CNN 的一个显著特征。局部性和深度的含义是什么？背后有什么原理？目前这尚是一个开放问题。本章尝试在卷积层与元胞自动机之间建立联系，从元胞自动机的动力学角度分析 CNN 局部性和深度的意义。

元胞自动机是一种基于局部规则的离散动力系统，它具有“自组织”的特性，能够从简单的规则中涌现复杂而奇妙的结构和行为。本章首先介绍元胞自动机的定义，接着在多层卷积层与元胞自动机之间建立联系。我们会看到，多层卷积层所执行的计算就是元胞自动机随时间的运行。接着，我们介绍元胞自动机的分类，以及不同类别的元胞自动机的行为。在讲解图灵可计算性的相关理论之后，我们介绍部分元胞自动机规则的图灵完备性。最后，本章从动力学吸引子和分岔的角度为 CNN 的分类能力和训练过程提供一些有趣的观察和推测。

所有这些内容，都是为理解“深度”这个深度学习核心概念提供一个新的视角。需要注意，本附录中的一些说法仅是作者的猜测，读者应该注意对其中未经验证的假说加以甄别和防范。欢迎有兴趣的读者对本章的一些猜测做实际的实验观察。

A.1 二维元胞自动机与 CNN

将二维平面划分成网格（lattice），每个小格子称为元胞（cell），元胞可取k个值$0, 1, \cdots, k-1$。元胞的值随离散的时刻 0, 1, 2, …发生变化。元胞在下一时刻的值取决于它的邻域内的元胞（含自己）在本时刻的值，这就是（二维）元胞自动机（cellular automata，CA）。邻域的大小任意，例如 3×3，5×5 等。如果邻域从中心向各方向延伸r个元胞，则称该邻域为r-邻域。例如大小为 3×3 的邻域是 1-邻域。网格上全体元胞的一组特定取值称为一个构型（configuration）。邻域内全体元胞的一组特定取值称为邻域构型。

二维 CA 的r-邻域包含$(2r+1)^2$个元胞，每个元胞有k种取值，所以二维 CA 的r-邻域共有$k^{(2r+1)^2}$种不同的邻域构型。根据当前邻域构型决定中心元胞下一时刻取值的规则是一个映射，它将$k^{(2r+1)^2}$种邻域构型映射到k个值。这样的规则（映射）共有$k^{k^{(2r+1)^2}}$种，每一种规则决定了一种r-邻域二维 CA，也就是说共有$k^{k^{(2r+1)^2}}$种r-邻域二维 CA。John Conway 设计的生命游戏（game of life）是k为 2，r为 1 的二维 CA，它采用如下规则。

- 若中心元胞值为 1 且邻域内除自己外少于两个元胞值为 1，则中心元胞下一时刻值为 0（人口过少）；
- 若中心元胞值为 1 且邻域内除自己有两或三个元胞值为 1，则中心元胞下一时刻值为 1（人口适中）；
- 若中心元胞值为 1 且邻域内除自己有超过三个元胞值为 1，则中心元胞下一时刻值为 0（人口过剩）；
- 若中心元胞值为 0 且邻域内正好有三个元胞值为 1，则中心元胞下一时刻值为 1（繁殖）。

生命游戏按照这个规则运行起来会产生各种结构，有的结构保持不变，有的结构循环变形，有的结构循环变形的同时在网格中运动，还有的结构不停生成并发射出新的小结构。图 A-1 展示了一种变形结构。

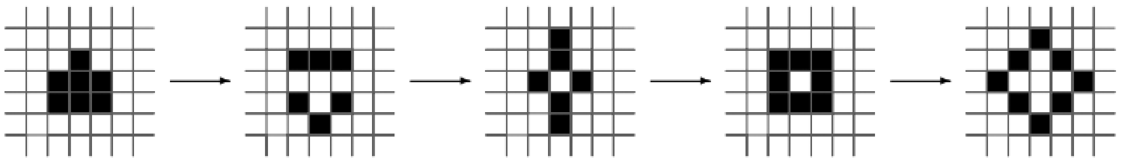


图 A-1　生命游戏的一种变形结构

二维 CA 的规则与卷积层执行的计算类似，它们都是根据邻域的值确定中心位置的值。例如这样的规则γ：若邻域内有 3 个（含）以上元胞的值为 1，则下一时刻中心元胞为 1，否则为 0。可以构造一个仿射加激活的函数实现这个规则，令向量$\boldsymbol{w}$的分量都为 1，将邻域内全体元胞的值列成向量$\boldsymbol{x}$，构造函数：

$$f(\boldsymbol{x})=\begin{cases}0, & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-3<0\\ 1, & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-3\geqslant 0\end{cases} \tag{A.1}$$

式（A.1）先对输入施加权值全为 1、偏置是–3 的仿射函数，之后施加阶跃函数。若将权值向量排成方阵，就是一个二维卷积核。以该方阵为卷积核、以–3 为偏置，以阶跃函数为激活函数，我们就构造了一个卷积层。这个卷积层对当前构型的输出，正是当前构型在规则γ下的变化。

如果二维网格不是无限延伸的，可将网格的对边连在一起，构成如图 A-2 所示的圆环面（torus），网格边缘元胞的邻域扩展到网格对侧。相应地，若卷积层采取翻转（wrap）填充方式，就可以模拟圆环面上的 CA。

单个卷积核无法模拟全部规则。以k为 2、r为 1 的 CA 为例，它的 2^9种邻域构型的每一种都是 9 维单位立方体的一个顶点，规则将这些顶点映射到 0 或 1，相当于将这些顶点分为两类。卷积核是一个线性模型，若某个规则是线性不可分的，则单个卷积核无法模拟它。

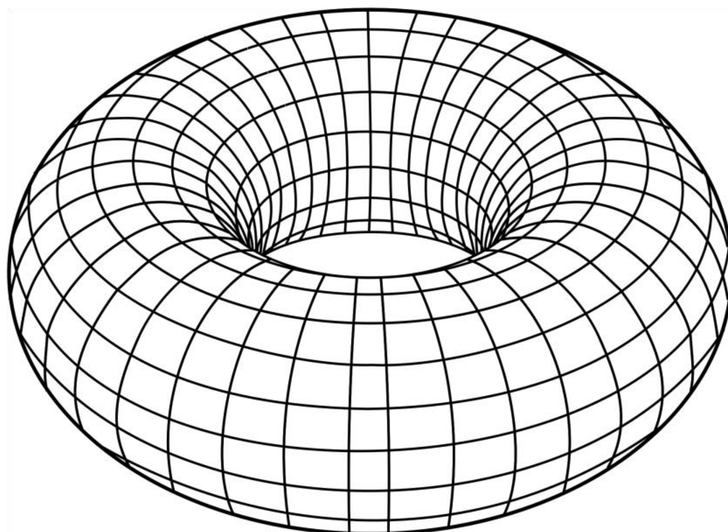


图 A-2　圆环面

可以使用多个卷积核模拟线性不可分规则，令每一个卷积核模拟一个线性可分规则，再将它们的结果线性组合，之后施加阶跃函数。这和把多个线性模型组合起来形成非线性分界面的原理相同。上述办法相当于在一个多卷积核卷积层之后连接一个 1×1 的单卷积核卷积层。上一章介绍的局部响应标准化（LRN）技术可以迫使同卷积层的不同卷积核形成差异，有利于它们合作模拟线性不可分规则。从模拟 CA 的角度看，1×1 卷积核和 LRN 技术具有这样的意义。

所以，用一个多卷积核卷积层后连 1×1 单卷积核卷积层（共两层），就可以模拟任何 CA 规则。为了论述简洁，后文我们说用卷积层模拟某规则，其实是指这样的复合层。把n个模拟规则γ的相同卷积层连接成一个深度为n的 CNN，以某个初始构型作为这个 CNN 的输入，计算该 CNN 的输出，这相当于一个 CA 在规则γ下从初始构型开始运行n步，如图 A-3 所示。

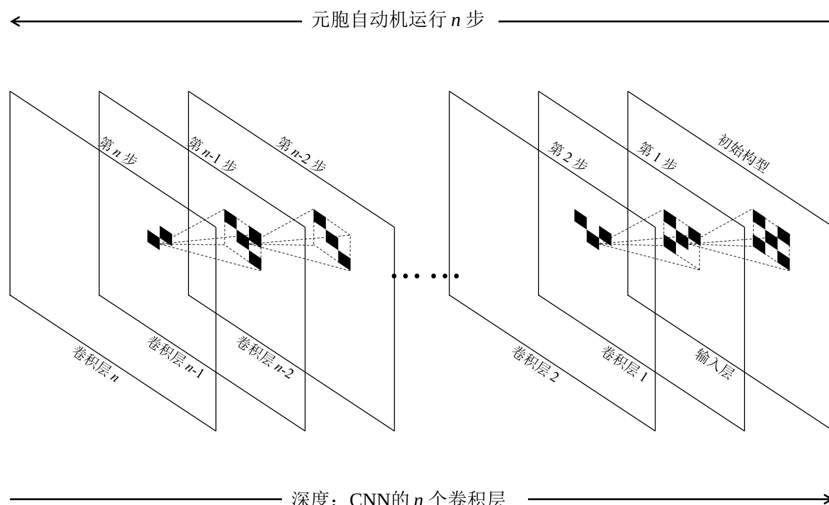


图 A-3　n个卷积层相当于元胞自动机运行n步

CA 和卷积层不限于二维，一维 CA 的网格是一个无限或有限长的条带。若条带长度有限，则我们可以将其视作首尾相接的圆环。一维r-邻域由向左右延伸各r个位置的$2r+1$个元胞组成。k种取值、r-邻域的一维 CA 共有k^{2r+1}种邻域构型和$k^{k^{2r+1}}$种规则。一维卷积层可以模拟一维 CA 规则，如图 A-4 所示。

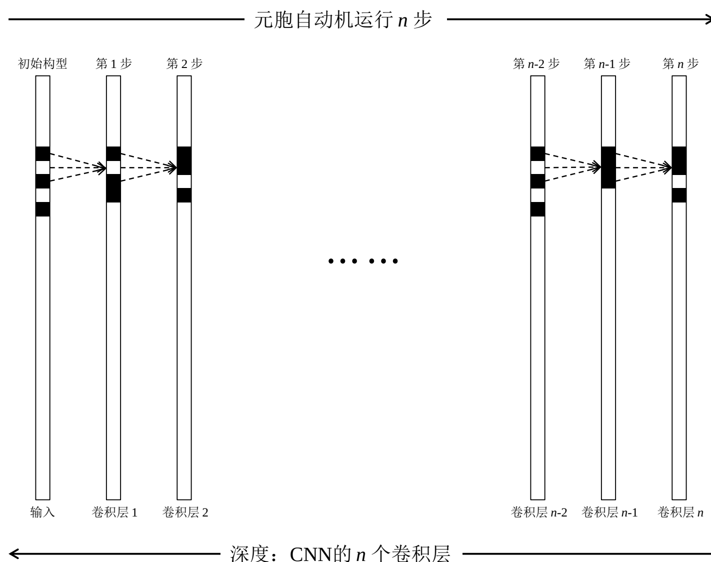


图 A-4　一维卷积层模拟一维 CA 的运行

注意，我们用来模拟 CA 的是一类很受限的 CNN，它们的所有卷积层都相同。上文说过任何 CA 都可以用 CNN 模拟，但是否任何 CNN 都可以被 CA 模拟呢？答案好像是否定的，因为普通 CNN 比我们用来模拟 CA 的这类受限的 CNN 要灵活得多。即使不考虑池化、全连接以及其他各种组件，至少普通 CNN 不要求所有卷积层都相同。但是在后文介绍了图灵完备性后，我们会知道其实 CA 是可以模拟普通 CNN 的。

我们的目的是探究深度的含义，我们可以只在这类受限的 CNN 下进行考察，如果能在受限的情况下稍稍洞悉深度的含义，这对我们理解普通 CNN 乃至其他深度模型都是有启发的。既然（受限的）CNN 的深度等价于它所模拟的 CA 的运行步数，那么观察 CA 的长期运行行为也许可以为理解深度提供一些洞见。在一维情况下方便更清晰地分析和展示 CA 的行为，所以下文以一维 CA 为例进行讲解。

A.2　元胞自动机的行为和分类

k为 2、r为 1 的一维 CA 称作基本元胞自动机（elementary cellular automata，ECA）。ECA 是最简单的元胞自动机，它有 $2^3 = 8$ 种邻域构型。ECA 的邻域构型是三维立方体的 8 个顶点，如图 A-5 所示。

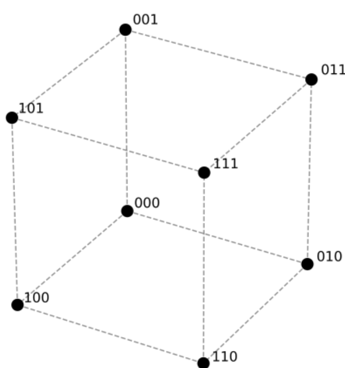


图 A-5　ECA 的 8 种邻域构型

将 ECA 的 8 种邻域构型写成二进制数：000、001、010、011、100、101、110、111，规则为每种邻域构型分配 0 或 1，例如表 A-1 所示的规则。

表 A-1　ECA 规则 110

邻域构型	111（7）	110（6）	101（5）	100（4）	011（3）	010（2）	001（1）	000（0）
值	0	1	1	0	1	1	1	0

表 A-1 中的规则可看作 8 位二进制数：01101110_2。该二进制数是十进制的110_{10}，所以这个规则被称为“规则 110”。每一个 8 位二进制数都对应一种 ECA 规则，共有 2^8 = 256 种规则。用对应的十进制数命名它们，就是规则 0~255。可以用 3 维单位立方体展示规则：白色顶点表示规则将该邻域构型映射到 0，黑色顶点表示映射到 1，如图 A-6 所示。可以看到规则 110 是线性不可分的。

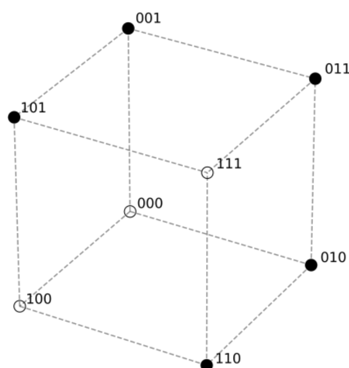


图 A-6　规则 110 的单位立方体

A.2.1　相空间、轨迹与吸引子

若网格条带长度为L，则 ECA 共有2^L种构型（不是邻域构型，而是整个网格条带的构型）。所有构型的集合称为相空间（phase space）。每一种构型都是相空间中的一个点，表示网格条带的一个状态。规则确定了当前构型在下一时刻的变化。从一个初始构型出发，规则决定了构型在相空间中演化的轨迹。ECA 是一个确定性的离散动力系统。

图 A-7 中的点表示网格条带长度为 12 的 ECA 的构型，共有 4096 种，箭头表示构型在规则 90 下的变化，箭头尾部的构型变化为箭头指向的构型。箭头连成的有向路径就是该 ECA 在规则 90 下所有可能的构型演化轨迹（trajectory）。

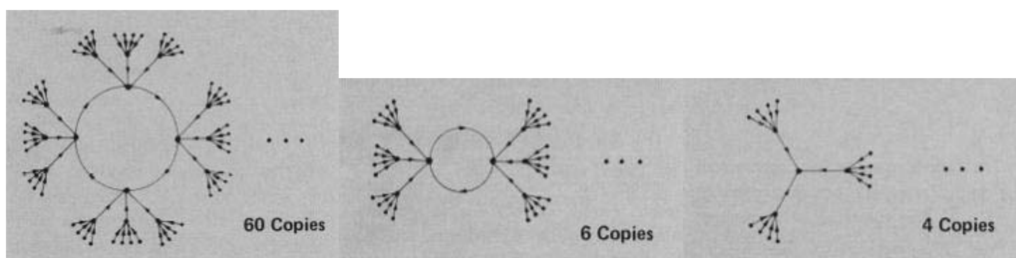


图 A-7　网格条带长度为 12 的 ECA 在规则 90 下的演化轨迹（来自参考文献[11]）

可以看出，全部构型被轨迹连接成若干互相隔离的簇，这些簇具有三种形态，每种形态都有多个副本。有些构型没有前导构型，如图中位于簇外围边缘的构型，即规则 90 不会把任何构型变成它们。除非作为初始构型，否则这样的构型不会在 ECA 的演化中出现。若它们是初始构型，规则 90 下的演化一旦启动，就再也不会回到它们。

该 ECA 的所有构型在规则 90 下最多经过两步就进入一个循环。例如顶部的簇的边缘构型经过两步进入一个由 4 个构型形成的循环。在规则 90 下，该 ECA 从任意初始构型出发经过一定时间的演化就会落入一个构型的子集，该子集以外的构型在之后的演化中不再出现，这个子集称为吸引子（attractor）。

吸引子若包含若干不连通的子集，则也可以将这些子集视为更细粒度的吸引子。该 ECA 在规则 90 下共有 60 + 6 + 4 = 70 个不可再分的细粒度吸引子。在这些吸引子中，有些是稳定的点，称为稳定点（stable point）；有些是若干构型的循环，称为极限环（limiting circle）。还有更复杂的吸引子，后文会做介绍。

A.2.2　不可逆性、信息擦除与熵减

从相空间轨迹图可以看出，规则 90 有可能将两条轨迹合并。若规则不是单射，则它有可能将不同构型映射到同一个构型，这就导致了轨迹的合并。在确定性动力系统中，每个构型的后续演化是确定的，但在会合并轨迹的规则下，每个构型的前导构型不确定。从 ECA 的当前构型可以确定后续任意步的运行路径，但无法确定之前的运行路径。用当前构型可以确定未来，但无法推测过去，我们称这样的动力系统是不可逆的（irreversible）。不可逆系统会丢失信息，或者说不可逆规则会擦除信息。

不可逆规则在运行中会将轨迹合并，导致一些构型不再出现。令初始构型服从全体构型上的均匀分布，在不可逆规则下，随着 ECA 的运行，某些构型出现的概率会降为零，那些概率没有降为零的构型就构成了吸引子。ECA 落到吸引子之前的时间称为瞬态（transient），瞬态是暂时的，ECA 最终会被吸引到吸引子上。如果规则是可逆的，相空间中的每一条轨迹都可以沿着正向和反向追溯，构型的概率分布不随运行变化，可逆系统的吸引子是全体构型。

构型的概率分布决定系统的熵，可逆系统的熵不变，而不可逆系统的熵会随着运行而降低。前文说过，不可逆规则的执行会擦除信息，根据 Landauer 原理：“任何不可逆过程，例如擦除信息或合并轨迹，都伴随着信息载体之外的环境的熵增。”即运行不可逆系统的物理装置会导致环境的熵增，也就是说它消耗能量。

不可逆程度不同的规则擦除信息的能力不同，导致它们的吸引子有不同的熵。稳定点的熵最低，极限环的概率在构成循环的多个构型间均匀分布，熵稍高。有些规则使 ECA 在瞬态展现出"自组织"行为：随着运行，构型能够形成一些复杂、持久且相互间有交互的结构，一种消耗能量维持自身秩序的结构，即普利高津的耗散结构（dissipative structure）。这就是薛定谔所说的"生命以负熵为食"(《生命是什么》)。

A.2.3　康托尔集与分形维数

若 ECA 的网格条带长度为 10，则它一共有 $2^{10} = 1024$ 种构型，令初始构型以等概率取这 1024 种构型之一，并运行规则 10。规则 10 是不可逆的，随着运行轨迹发生合并，部分构型的概率降为零，如图 A-8 所示。

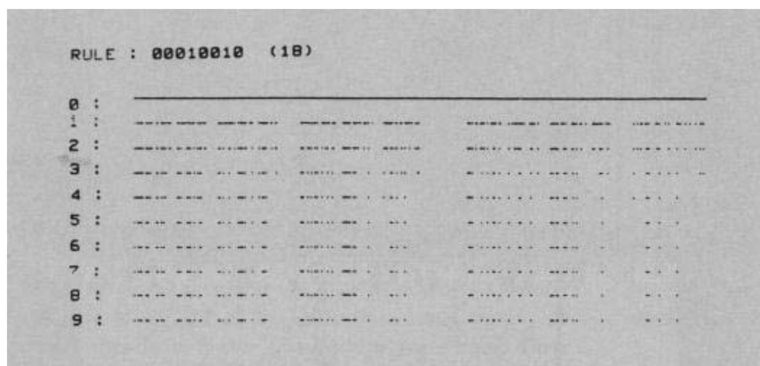


图 A-8　构型概率随着规则 10 运行而变化（来自参考文献[11]）

图中将 1024 种构型排成一行，每一行是一个时间步。行内每一个点表示对应的构型的概率是否非零：黑色表示构型概率非零，无色表示构型概率为零。图中的一列就是某一构型的概率变化。随着 ECA 运行，有些构型的概率降为零，有些构型的概率保持非零。最终概率非零的构型组成规则 10 的吸引子。

长度无限的网格条带的构型是无限长度的 0-1 串，若将无限长度的 0-1 串作为[0, 1]区间内二进制实数的小数部分，则构型可与[0, 1]区间内的实数一一对应。规则的吸引子有可能包含有穷个构型，也有可能包含无穷个构型。有些规则的吸引子构成一个康托尔集（Contor set）。

最简单的三分康托尔集是这样构造的：将[0, 1]区间内的实数的中间三分之一，即$\left(\frac{1}{3},\frac{2}{3}\right)$抠除，只保留左右两个三分段，然后再对剩下的这两个三分段执行同样的抠除，此过程无限进行下去，得到的集合就是三分康托尔集，如图 A-9 所示。

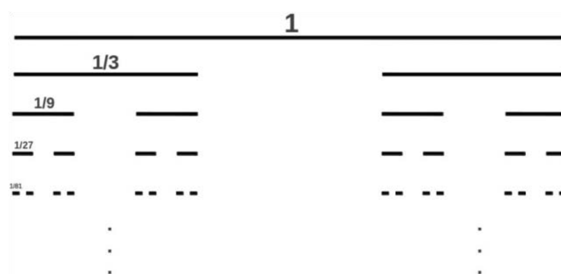


图 A-9　三分康托尔集

三分康托尔集和[0, 1]区间内的实数可以一一对应。任意[0, 1]区间内的实数写成二进制都是小数点后一串无穷长度的 0-1 串，例如0.110110010 ...，以 0-1 串的每一位为指示：若遇 0 则选择左侧三分段；若遇 1 则选择右侧三分段，这样，每一个无穷 0-1 串就确定了一个三分康托尔集中的数。反过来，根据一个数在每个层次位于左或右哪个三分段，就决定了唯一的一个无穷 0-1 串。

三分康托尔集具有自相似性（self-similarity）：将任意一个层次上的一个三分段放大，会发现它具有与整个三分康托尔集同样的结构。三分康托尔集是一个分形（fractal），它具有分数维数（fractal dimension）。

分数维数有多种定义，可以用自相似性定义分数维数。三分康托尔集的每个三分段含有 2 个尺度是自身$\frac{1}{3}$，且与自身相同的段，于是它的分数维数定义为$\log_3^2 \approx 0.63$。还有其他康托尔集，比如将中间二分之一段抠除。这样形成的康托尔集称为二分康托尔集。二分康托尔集的每个段含有 2 个尺度是自身$\frac{1}{4}$，且与自身相同的段，所以它的分数维数是$\log_4^2 = 0.5$。基于同样的理由，二分康托尔集也是无穷的，且可与[0, 1]区间内的实数一一映射，所以二分和三分康托尔集具有相同的基数（cardinal number），但是二分康托尔集的分数维数小于三分康托尔集。

还有另一种分数维数的定义，如果最少需要$N(\varepsilon)$个长度为ε的线段才能覆盖康托尔集，则该康托尔集的分数维数是：

$$d = \lim_{\varepsilon\to 0}\frac{\log^{N(\varepsilon)}}{\log^{\frac{1}{\varepsilon}}} \tag{A.2}$$

这种分数维数称为豪斯多夫维数。考察三分康托尔集的豪斯多夫维数：它在第n个层次上可以被2^n个长度为3^{-n}的线段覆盖，即$N = 2^n$，豪斯多夫维数是：

$$d = \lim_{\varepsilon\to 0}\frac{\log^{2^n}}{\log^{3^n}} = \log_3^2 \tag{A.3}$$

三分康托尔集的豪斯多夫维数与基于自相似的维数相同。[0, 1]线段可以被$\frac{1}{\varepsilon}$个长度为ε的线段

覆盖，它的豪斯多夫维数是 1。由有穷个数组成的集合可以被有穷条任意短的线段覆盖，所以它的豪斯多夫维数是零。豪斯多夫维数是普通维数的扩展，以下我们一律采用豪斯多夫维数。

若规则是可逆的，随着运行，ECA 将遍历所有构型，每一个构型都保持非零概率，称这样的规则具有可遍历性（ergodicity）。可逆规则运行时不丢失信息，其吸引子是全体构型，维数为 1。若吸引子只包含有穷构型，则它的维数是零。这两种吸引子具有整数维数，它们不是分形。

有些规则的吸引子是分形，具有分数维数。规则的不可逆性越强，则轨迹合并越严重，执行过程中信息擦除越多，吸引子的分数维数越小。若不可逆性强到一定程度，则吸引子的维数降为零，只包含有穷构型。规则的不可逆性越强的，将导致构型的概率分布越集中，吸引子的熵越低。

A.2.4　4 类规则与混沌边缘

Stephen Wolfram 研究了 ECA 各种规则的行为，他将规则分为 4 类。4 类规则的不可逆程度不同，它们的吸引子有不同的分数维数，表现出不同的行为。本节用例子分别介绍这 4 个类别。

1. 第一类规则

规则 160 属于第一类规则，它的运行如图 A-10 所示。初始构型随机，网格条带长度 800，运行 400 步，可以看到在规则 160 下，ECA 迅速演化成全零构型，落了片白茫茫大地真干净。这就是第一类规则的行为特点，它们最终将演化为全 0 或全 1 构型。第一类规则的吸引子只包含一个构型，维数和熵都为零。第一类规则具有高度规律性，它们迅速消除了系统的随机性。第一类规则是信息擦除能力最强、轨迹合并最严重、消耗能量最多、最终的熵最低的系统。

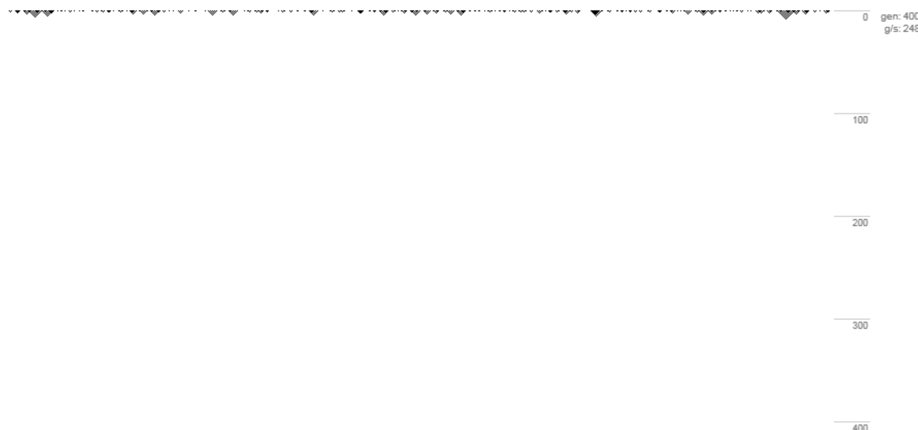


图 A-10　第一类规则的例子：规则 160

2. 第二类规则

规则 108 属于第二类规则，它的运行如图 A-11 所示（运行 800 步）。第二类规则形成被空白（零值）隔开的一些区域，这些区域或者稳定在一种局部构型上，或者在若干局部构型之间循环，形成类似梯子格的形态。

第二类规则的吸引子包含有穷构型，维数为零。在第二类规则下，每个元胞的最终值只受到它的有限邻域内元胞的初始值影响，也就是说，一个元胞取不同初始值的影响不会扩散到整个网格，而只能产生局部影响。改变一个元胞的初始值，后续的演化只受到有限的影响。若两个初始构型差异极小，则它们后续的运行也保持较小的差异。第二类规则对初始条件不敏感。

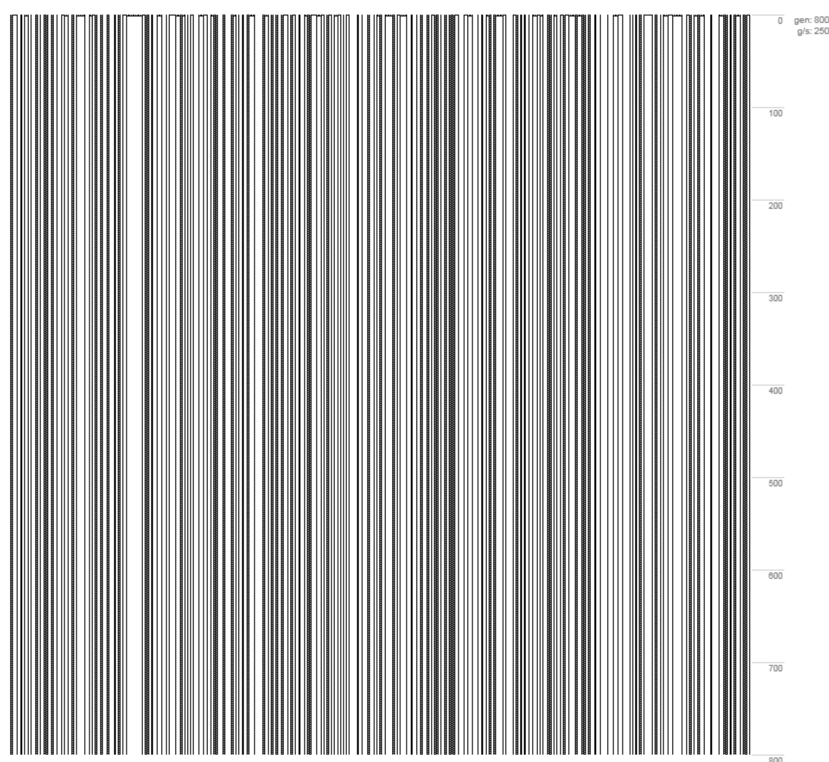


图 A-11　第二类规则的例子：规则 108

3. 第三类规则

规则 126 属于第三类规则，它的运行如图 A-12 所示（运行 800 步）。规则 126 的行为很复杂，运行中构型始终保持较大的随机性。可以看到有一些连续的零值（空白）突然形成，但马上被周

围的随机区域吞噬，形成图中大大小小的倒三角。这类规则的随机性过高，任何有序结构都不能持久，系统保持较高的熵。第三类规则接近可逆系统，吸引子的维数接近 1。

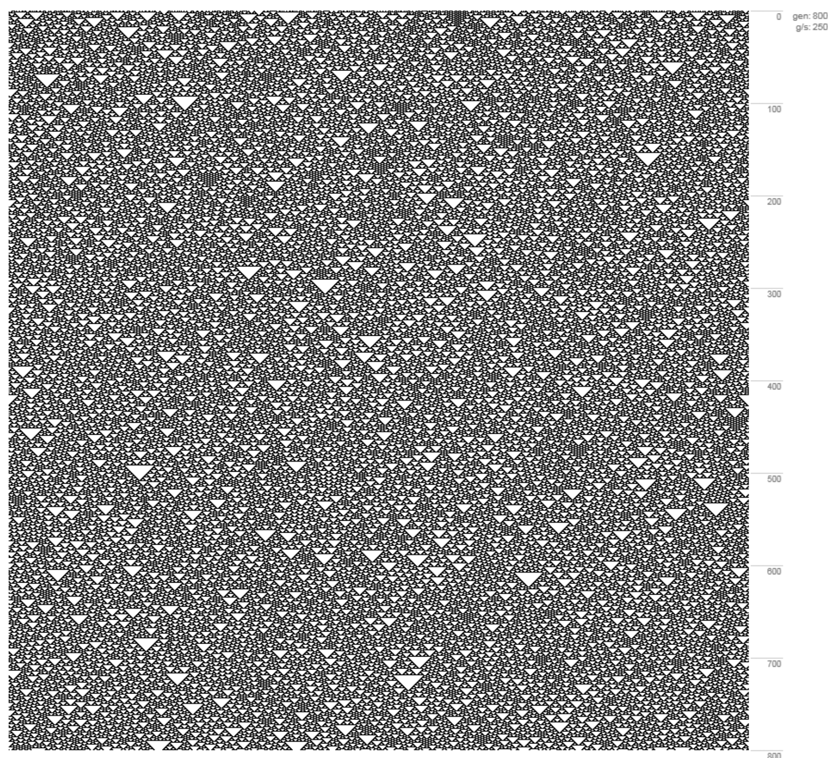


图 A-12　第三类规则的例子：规则 126

如果初始构型只有一个元胞的值为 1，其余元胞的值为 0，则规则 126 下的运行图如图 A-13 所示，该运行图也是一个分形，具有自相似性。在第三类规则下，一个元胞的影响匀速扩散到无穷远处。反过来说，只要时间够长，一个元胞的值会受到任意远处的元胞的初始值的影响。初始构型的微小变化在一定时间后会导致运行轨迹的巨大差异。第三类规则表现出对初始条件的敏感性，它们的行为是混沌的（chaotic）。

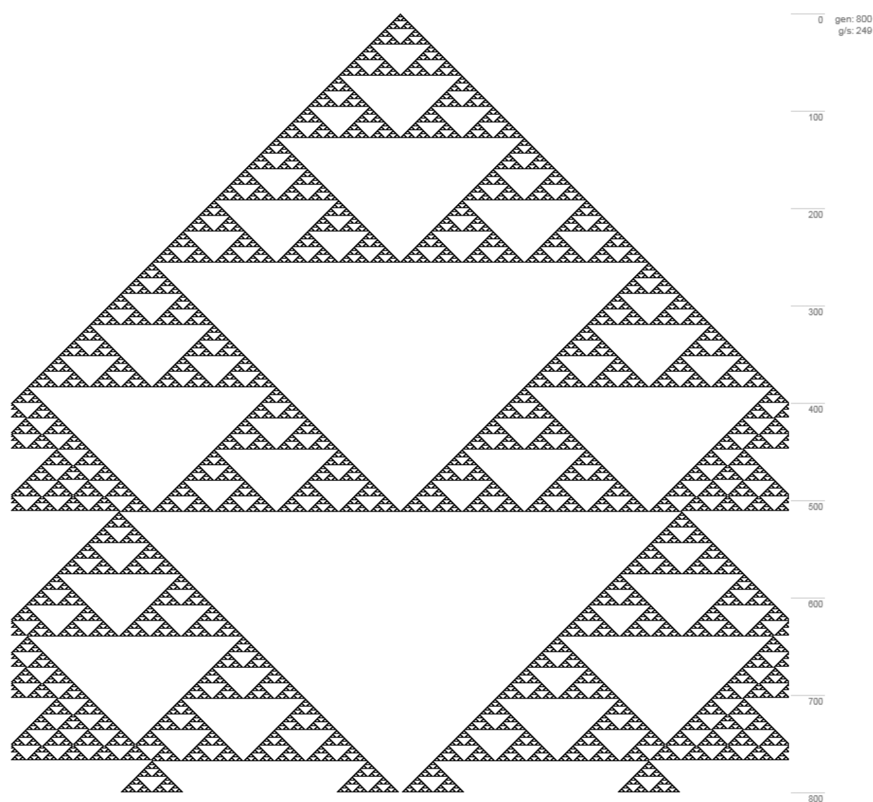


图 A-13　从只有一个非零元胞的初始构型开始按规则 126 运行

第三类规则具有分形吸引子，分形吸引子又称奇异吸引子（strange attractor）或混沌吸引子（chaotic attractor）。第三类规则对应连续动力系统中的混沌系统，例如洛伦兹（Lorenz）系统：

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y-x) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y \\ \dfrac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z \end{cases} \tag{A.4}$$

(x,y,z)是点在三维相空间中的位置。点在相空间中某个位置的速度向量$\left(\frac{\mathrm{d}x}{\mathrm{d}t},\frac{\mathrm{d}y}{\mathrm{d}t},\frac{\mathrm{d}z}{\mathrm{d}t}\right)$由方程组（A.4）确定，$\sigma$、$\rho$和$\beta$是三个预设参数。点根据这个简单的非线性微分方程组在相空间中运动，瞬态过后被吸引到洛伦兹吸引子上。洛伦兹吸引子是奇异吸引子，如图 A-14 所示，它是一个三维空间中的分形，豪斯多夫维数是 2.332。

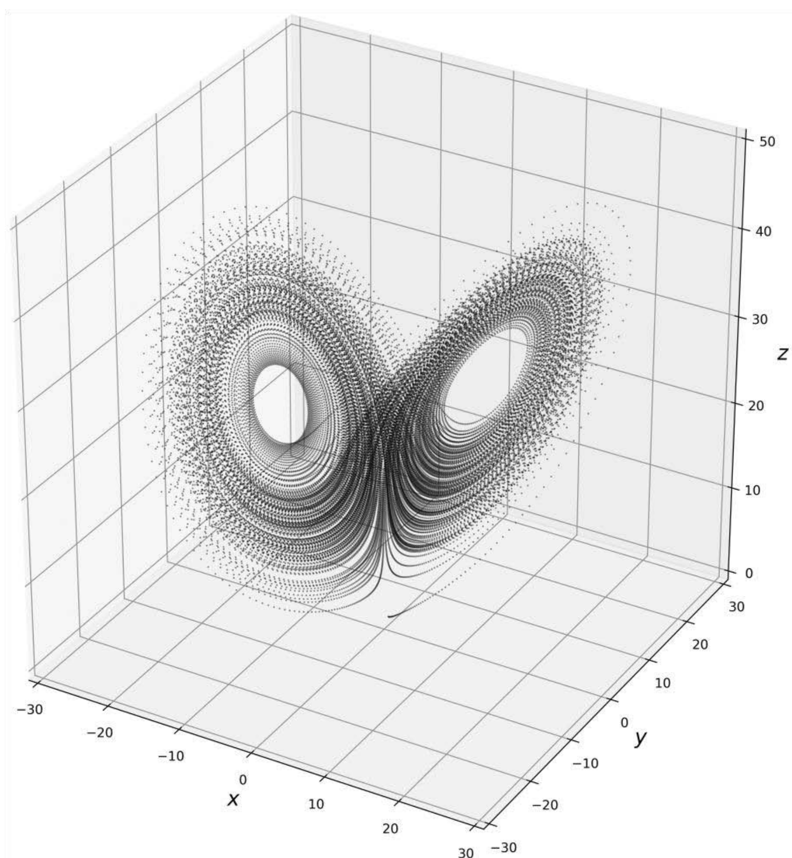


图 A-14　洛伦兹吸引子

若两个点在相空间中的初始位置存在微小差异，按洛伦兹方程运动一段时间后，它们的位置将存在巨大差异。混沌系统具有长期不可预测性——初始状态的微小差异随着时间以指数速率增长，这就是所谓“蝴蝶效应”。“蝴蝶效应”一词常被误用：并非蝴蝶翅膀扇出的微小气流最终演变成了风暴，而是蝴蝶扇或者不扇翅膀所造成的初始状态的微小差异，导致最终结果产生巨大差异——有或者没有风暴。

初始位置的坐标可能具有无穷精度，例如无理数，所以混沌表明：即便对于确定性的系统，长期的预测也是不可能的。混沌系统正如怀特海所说：“你可以期待明天太阳从东方升起，但你不知道风将从何方吹来。”(《科学与近代世界》)

第三类规则下的 ECA 是混沌系统，它展现出对初始条件的极端敏感性和长期行为不可预测性：无论两个初始构型多么相似，一定时间后它们后续的演化将存在指数级的差异。混沌行为阻

止持久复杂结构的形成。自然界存在疑似第三类规则的现象，例如贝壳随着生长形成的色素沉积图案，如图 A-15 所示。

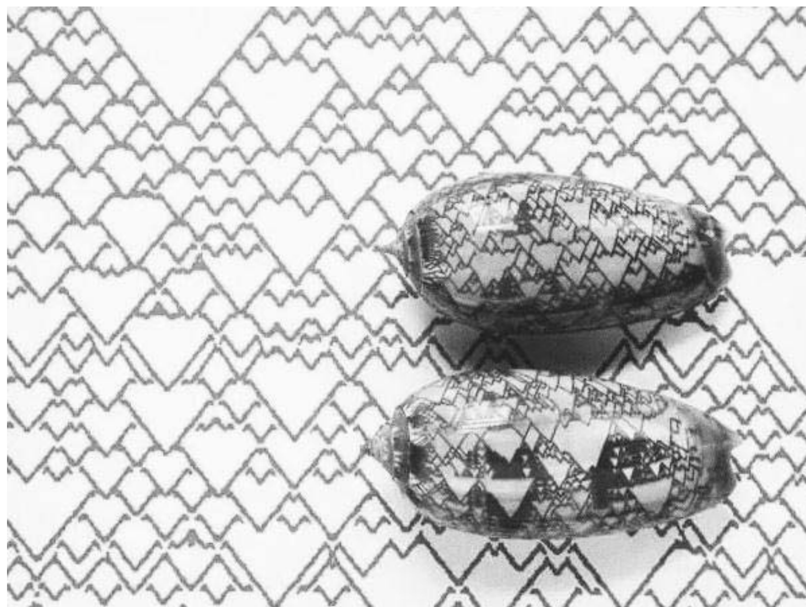


图 A-15　贝壳的色素沉积

4. 第四类规则

规则 110 属于第四类规则，它的运行如图 A-16 所示（运行 1200 步）。第四类规则的吸引子的维数介于第二类和第三类规则之间。第四类规则下，ECA 在瞬态能够自发形成一些结构，这些结构比第二类规则的结构更复杂，比第三类规则的空白区域更持久。这些结构以复杂的方式改变自身形态，还可以沿着网格运动，结构之间还可以发生复杂的交互。它们是生命游戏中那些复杂结构的一维对应物。

从随机中自发形成并维持复杂的结构，这个现象称为“自组织”（self organization）。这些复杂结构是耗散结构，耗散结构与外界发生能量和物质交换，在一定时间内维持自身的秩序。耗散结构的产生需要一定的环境条件，以 ECA 来说，完全可逆的规则是熵最大的平衡态，第三类规则接近平衡态，熵仍然过大，第一、二类规则距离平衡态过远，缺乏足够的随机性。耗散结构只能产生于第二类规则，即离开平衡态不太远的混沌边缘（edge of chaos）。

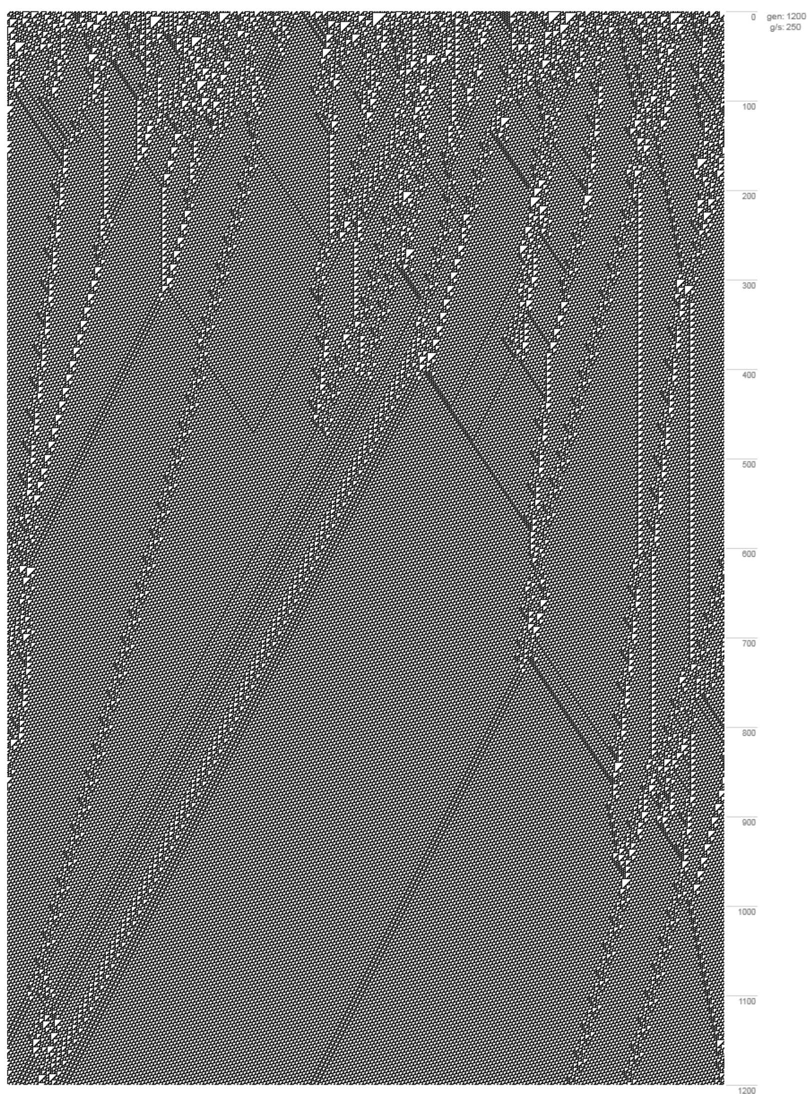


图 A-16　第四类规则的例子：规则 110

A.3　图灵机与可计算性

本节介绍图灵机和可计算性等概念，并考察元胞自动机能力的上限以及它能否达到这个上限。图灵机（Turing machine）是 Alan Turing 提出一种假想的机器，它是一种形式化定义“计算”的模型。想一下在我们草稿纸上计算一个乘法的过程：首先将“竖式”写在纸上，然后，从某个位置开始，根据看到的符号按照规则（乘法表）写下新的符号，之后移动视线到纸上的其他位置，

同时大脑还需要记忆一些信息：当前是否存在进位，进到高位的数字是什么，等等。进行几轮动作后计算过程结束，在纸上留下计算结果。图灵机模型就是要模拟类似的过程。

A.3.1　图灵机

将一台图灵机记为M，M具有一个有穷状态集S，任意时刻M处于S中的某个状态：$s \in S$。S中有一个状态称作开始状态，记为$s_{\text{start}} \in S$。S有一个子集称作接受状态集，记为$S_{\text{accept}} \subset S$。$S_{\text{accept}}$中的状态称为接受状态。$S$还有一个子集称作拒绝状态集，记为$S_{\text{reject}} \subset S$。$S_{\text{reject}}$中的状态称为拒绝状态。$S_{\text{accept}}$和$S_{\text{reject}}$不相交，即$S_{\text{accept}} \cap S_{\text{reject}} = \emptyset$。也就是说一个状态不能既是接受状态又是拒绝状态。

M有一个有穷的字符集合Σ，例如$\Sigma = \{\alpha, \beta, \gamma\}$或$\Sigma = \{0,1\}$。令$\Sigma^*$是集合，它的元素是所有由有穷个$\Sigma$中的字符连接成的字符串。对于$\Sigma = \{\alpha, \beta, \gamma\}$来说，所有由有穷个$\alpha$、$\beta$或$\gamma$连接成的字符串$\omega$都属于$\Sigma^*$，例如$\omega = \alpha\alpha\beta\gamma$。字符串$\omega$包含的字符个数是它的长度。长度为零的字符串称为空字符串（null string），空字符串也属于Σ^*。

M有一个无限长的带（tape），带被分成一个个单元格（cell），每个单元格上可以写一个字符。M有一个读写头，总是位于带的某个单元格之上。读写头可以对当前单元格进行读和写，还可以沿着带左右移动，但一次只能移动一个单元格。允许在带上出现的全部字符构成“带字符集”，记为Γ。Γ包含Σ：$\Gamma \supset \Sigma$。Γ还可以包含Σ中没有的字符，这些字符不能用来构造输入字符串ω，但可以被M在带上读和写。Γ至少应包含一个Σ中没有的字符——空白字符（注意不是空字符串，而是一个表示空白的字符）。

在运行的每一步，M根据当前状态和读写头下的字符，擦除当前单元格的旧字符并写下某个新字符，将读写头向左或右移动一个单元格，进入某个新状态（新状态也可以就是本来的状态）。决定M如何动作的规则就是M的转移函数$\delta: S \times \Gamma \to S \times \Gamma \times \{\text{left}, \text{right}\}$，$\times$表示笛卡尔积，即由多个集合所有元素形成的所有可能元组。若$\delta(s_1, \alpha) = (s_5, \beta, \text{right})$，则当$M$处于状态$s_1$且读写头下单元格的字符是$\alpha$时，擦掉$\alpha$写下$\beta$，读写头向右（right）移动一格，进入状态$s_5$。7-元组$M = (S, s_{\text{start}}, S_{\text{accept}}, S_{\text{reject}}, \Sigma, \Gamma, \delta)$就定义了一台图灵机$M$，不同 7-元组定义不同的图灵机。

要启动M，首先将输入字符串$\omega \in \Sigma^*$写在带上的任意位置，带的其余单元格都是空白字符。令M的读写头对准ω的第一个字符，并让M处于开始状态s_{start}，这时M就开始一步一步地运行：读字符、覆写字符、移动读写头、进入新状态，然后再重复……直到M进入某个接受状态或拒绝状态，这时M停机。如果进入的是接受状态，则M接受输入字符串ω；如果进入的是拒绝状态，

则M拒绝ω。除这两种情况外还有第三种情况：M永远不停机——它既不进入接受状态也不进入拒绝状态，而是一直运行下去。后两种情况合称M不接受ω。注意“接受、拒绝、不接受”三者的区别：对于某输入字符串$\omega \in \Sigma^*$，M要么接受它，要么不接受它。如果M不接受ω，有可能是M停机并拒绝ω，也有可能是M不停机。图灵机的示意图如图 A-17 所示。

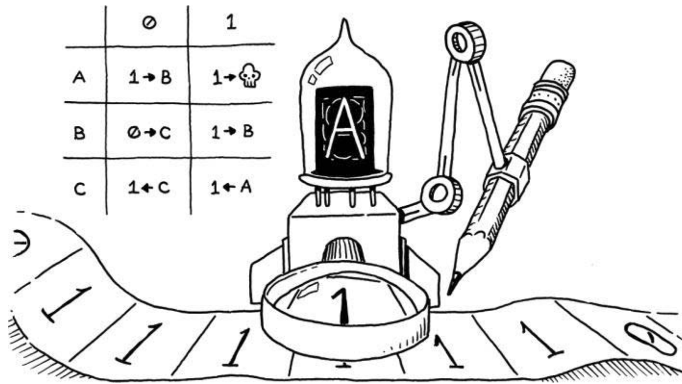


图 A-17　图灵机示意图

一个语言L（language）是Σ^*的一个子集，即$L \subset \Sigma^*$。空集$\emptyset$是一个语言——空语言。Σ^*本身也是一个语言，它包含字符集Σ上所有可能的字符串。可被图灵机M接受的所有字符串是一个语言，称为被M识别的语言，记作L_M。对于$\omega \in L_M$，若将ω作为M的输入，M将进入接受状态而停机；对于$\omega \notin L_M$，若将ω作为M的输入，M将进入拒绝状态而停机或者永不停机，即M不接受ω。

一台图灵机M可以被编码为一个字符串$\widetilde{M}$。$\widetilde{M}$以某种格式编码M的 7-元组。M的状态集S只包含有穷个状态，其中哪个是开始状态、哪些属于接受状态以及哪些属于拒绝状态都容易标记。输入字符集Σ和带字符集Γ也都是有穷集合。规则δ的定义域和值域都是有穷离散集合。所以只要定义好格式，M就可以用一个字符串$\widetilde{M}$编码。如下所示就是作者按照自定义的一种格式编码的一台图灵机：

```
0,1,+
0,1,+,=,.
1#0:0:L:2|1:1:L:2|+:+:L:2|=:=:L:2|.:.:L:2|::R:q#1#0#0
2#0:0:R:q|1:1:R:q|+:+:R:q|=:=:R:q|.:.:R:q|:=:R:3#0#0#0
3#0:0:R:3|1:1:R:3|+:+:R:3|=:=:R:3|.:.:R:3|::L:5#0#0#0
4#0:0:R:4|1:1:R:4|+:+:R:4|=:=:R:4|.:.:R:4|::L:f#0#0#0
5#0:.:L:7|1:.:L:8|+:+:L:6|=:=:R:q|.:.:L:5|::R:q#0#0#0
6#0:.:L:b|1:.:L:c|+:+:R:q|=:=:R:s|.:.:L:6|::R:q#0#0#0
7#0:0:L:7|1:1:L:7|+:+:L:9|=:=:R:q|.:.:R:q|::R:q#0#0#0
8#0:0:L:8|1:1:L:8|+:+:L:a|=:=:R:q|.:.:R:q|::R:q#0#0#0
```

```
9#0:.:L:b|1:.:L:c|+:+:R:q|=:=:L:b|.:.:L:9|::R:q#0#0#0
a#0:.:L:d|1:.:L:e|+:+:R:q|=:=:L:d|.:.:L:a|::R:q#0#0#0
b#0:0:L:b|1:1:L:b|+:+:R:q|=:=:L:b|.:.:R:q|:0:R:3#0#0#0
c#0:0:L:c|1:1:L:c|+:+:R:q|=:=:L:c|.:.:R:q|:1:R:3#0#0#0
d#0:0:L:d|1:1:L:d|+:+:R:q|=:=:L:d|.:.:R:q|:1:R:3#0#0#0
e#0:0:L:e|1:1:L:e|+:+:R:q|=:=:L:e|.:.:R:q|:0:R:4#0#0#0
f#0:.:L:h|1:.:L:i|+:+:L:g|=:=:R:q|.:.:L:f|::R:q#0#0#0
g#0:.:L:l|1:.:L:m|+:+:R:q|=:=:L:p|.:.:L:g|::R:q#0#0#0
h#0:0:L:h|1:1:L:h|+:+:L:j|=:=:R:q|.:.:R:q|::R:q#0#0#0
i#0:0:L:i|1:1:L:i|+:+:L:k|=:=:R:q|.:.:R:q|::R:q#0#0#0
j#0:.:L:l|1:.:L:m|+:+:R:q|=:=:L:l|.:.:L:j|::R:q#0#0#0
k#0:.:L:n|1:.:L:o|+:+:R:q|=:=:L:n|.:.:L:k|::R:q#0#0#0
l#0:0:L:l|1:1:L:l|+:+:R:q|=:=:L:l|.:.:R:q|:1:R:3#0#0#0
m#0:0:L:m|1:1:L:m|+:+:R:q|=:=:L:m|.:.:R:q|:0:R:4#0#0#0
n#0:0:L:n|1:1:L:n|+:+:R:q|=:=:L:n|.:.:R:q|:0:R:4#0#0#0
o#0:0:L:o|1:1:L:o|+:+:R:q|=:=:L:o|.:.:R:q|:1:R:4#0#0#0
p#0:0:L:p|1:1:L:p|+:+:R:q|=:=:R:q|.:.:R:q|:1:R:s#0#0#0
q#0:0:R:q|1:1:R:q|+:+:R:q|=:=:R:q|.:.:R:q|::R:q#0#0#1
r#0:0:R:r|1:1:R:r|+:+:R:r|=:=:R:r|.:.:R:r|::R:r#0#1#0
s#0:0:R:s|1:1:R:s|+:+:R:s|=:=:R:s|.:.:R:s|::L:t#0#0#0
t#0::L:t|1::L:t|+::L:t|=::L:r|.::L:t|::R:q#0#0#0
```

编码的第一行列出了输入字符集Σ中的字符：0、1 和+。逗号不算，它是编码的分隔符。第二行列出了带字符集Γ中的字符：0、1、+、=和.，默认用空格为空白字符。从第三行开始编码状态集和规则。每行表示S中的一个状态，用#分隔成 5 个字段，第 1 字段是状态的 id；第 3 字段用 0/1 标识该状态是否是开始状态；第 4 字段标识该状态是否是接受状态；第 5 字段标识该状态是否是拒绝状态。第 2 字段描述规则δ。δ的描述用|分隔成段，每一段再用:分隔，表示若M处于该状态且读到什么字符时写下什么字符、向什么方向（L/R）移动读写头，并进入哪个新状态。

比如其中第一行描述状态 1，它是开始状态，不是接受状态也不是拒绝状态。该图灵机若处于状态 1 且读到字符 0 时，覆写下字符 0，向左移动读写头并进入状态 2。读者可以根据该编码推断一下这台图灵机的功能。

每一台图灵机由 7-元组确定，执行某种特殊的计算。可以构建一种通用图灵机（universal Turing machine，UTM）。UTM的输入由某台图灵机M的编码$\widetilde{M}$和某个字符串ω连接而成。UTM从$\widetilde{M}$中解析出M的行为，并模拟M在输入ω上的动作，最终产生和M一样的结果：接受、拒绝或永不停机。UTM能够执行任何特殊图灵机的功能，我们使用的可编程计算机是UTM的一种实现，$\widetilde{M}$相当于程序，ω相当于程序的输入。Roger Penrose 在《皇帝新脑》中构建了一个UTM并将编码印在了书页中，如图 A-18 所示。

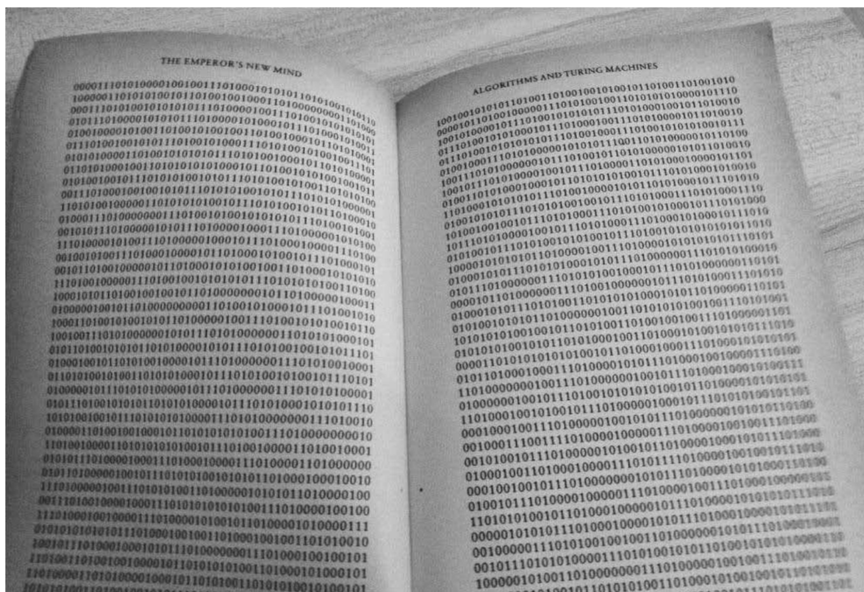


图 A-18　Roger Penrose 构建通用图灵机的编码

A.3.2　图灵–邱奇论题与图灵完备

图灵机不仅仅模拟“计算”，实际它定义“计算”。若要定义计算，当然首先得要求图灵机能执行人们常识中的计算，例如 $2\times3=6$。如果将“10×11”（即二进制的 2×3）作为输入字符串提供给一台图灵机，这台图灵机经过一系列执行后停机，并在带子上留下字符串“110”（二进制的 6），它就计算了 $2\times3=6$。上一节举例的那台图灵机的功能是计算二进制加法。

一种计算模型可以被另一种计算模型模拟，是指第一种计算模型能够执行的每一种操作，都可以用第二种计算模型的操作在常数步数内实现。这种情况称第一种计算模型的计算能力弱于第二种计算模型。迄今为止，人类构造的所有计算模型都能被图灵机模拟。任何计算装置，如算盘、智能手机、笔记本、超级计算机等，都不能超越图灵机的计算能力，这就是“图灵–邱奇论题”（Turing-Church thesis）。

“图灵–邱奇论题”是一个没有得到证明的假说，但是越来越多的验证使人们越来越确信这个假说是真的。有些模型的计算能力弱于图灵机，即图灵机可模拟它们，但它们无法模拟图灵机。有些模型等价于图灵机，它们与图灵机可以互相模拟。称计算能力等价于图灵机的模型是“图灵完备”（Turing-complete）的，例如冯・诺依曼结构计算机、邱奇的λ演算等。后文我们会看到部分 ECA 规则是图灵完备的。

A.3.3　递归语言与可计算性

对任何输入字符串都能够停机的图灵机称为判定器（decider）。对于任意输入字符串，判定器最终一定进入接受状态或拒绝状态，不会永远运行下去。有一类语言能够被判定器识别，这类语言称为递归语言（recursive language）。递归语言是语言的集合，对于该集合中的每一个语言，都存在一个判定器识别它。那么是否存在递归语言集合之外的语言，即无法被判定器识别的语言？答案是肯定的。设想下面的语言：

$$L_{\text{univ}} = \left\{\langle\widetilde{M},\omega\rangle \mid \omega \in L_M\right\} \tag{A.5}$$

其中，语言L_{univ}的元素是形如$\langle\widetilde{M},\omega\rangle$的字符串。$\langle\widetilde{M},\omega\rangle$表示把两个字符串$\widetilde{M}$和$\omega$用某种分隔符连接起来。$\widetilde{M}$是某台图灵机$M$的编码，$\omega$是某个能被$M$接受的字符串，即$\omega \in L_M$。也就是说，凡属于$L_{\text{univ}}$的字符串都是由两个字符串连接而成：第一个是某台图灵机M的编码$\widetilde{M}$，第二个是某个属于语言L_M的字符串ω。如果输入字符串格式不合法，即不是某个$\widetilde{M}$连接某个ω，则该字符串不属于L_{univ}。如果格式合法但是$\omega \notin L_M$，则该$\langle\widetilde{M},\omega\rangle$也不属于$L_{\text{univ}}$。

L_{univ}称作“通用语言”，因为它能告诉我们任何一台图灵机是否接受任何一个字符串。若一台图灵机的编码与一个字符串的连接属于L_{univ}，则表示该图灵机接受该字符串，否则该图灵机不接受该字符串。

现在假设L_{univ}属于递归语言，则存在一个判定器能够识别它，记为H。当$\langle\widetilde{M},\omega\rangle \in L_{\text{univ}}$，即当$\langle\widetilde{M},\omega\rangle$格式合法且$\omega \in L_M$时，$H$接受$\langle\widetilde{M},\omega\rangle$；当$\langle\widetilde{M},\omega\rangle \notin L_{\text{univ}}$，即$\langle\widetilde{M},\omega\rangle$格式不合法或$\omega \notin L_M$时，$H$拒绝$\langle\widetilde{M},\omega\rangle$。对$H$稍加改造，把它的接受状态和拒绝状态互换身份，得到新的判定器H'。于是当$\langle\widetilde{M},\omega\rangle \in L_{\text{univ}}$时，$H'$拒绝$\langle\widetilde{M},\omega\rangle$；当$\langle\widetilde{M},\omega\rangle \notin L_{\text{univ}}$时，$H'$接受$\langle\widetilde{M},\omega\rangle$。

接着再对H'做进一步改造，得到H''。H''的输入字符串是一台图灵机M的编码$\widetilde{M}$。H''首先构造字符串$\langle\widetilde{M},\widetilde{M}\rangle$，注意$\omega$的位置上现在是$\widetilde{M}$。之后$H''$在$\langle\widetilde{M},\widetilde{M}\rangle$上模拟$H'$的行为，那么$H''$的行为就是：若$\langle\widetilde{M},\widetilde{M}\rangle \in L_{\text{univ}}$，则$H''$拒绝$\widetilde{M}$；若$\langle\widetilde{M},\widetilde{M}\rangle \notin L_{\text{univ}}$，则$H''$接受$\widetilde{M}$。

关键之处到了。如果把H''自己的编码$\widetilde{H''}$输入给H''会发生什么？当$\langle\widetilde{H''},\widetilde{H''}\rangle \in L_{\text{univ}}$，即当$H''$接受$\widetilde{H''}$时，$H''$拒绝$\widetilde{H''}$；当$\langle\widetilde{H''},\widetilde{H''}\rangle \notin L_{\text{univ}}$，即当$H''$不接受$\widetilde{H''}$时，$H''$接受$\widetilde{H''}$。这里产生了悖论，所以结论只能是：最初的判定器$H$不可能存在。不存在判定器能够识别$L_{\text{univ}}$，即$L_{\text{univ}}$不属于递归语言。

一个非递归语言就是一个不可计算的问题、一个超出了计算机能力的问题、一个不能被任何算法解决的问题。波斯特对应问题（Post correspondence problem）就是一个不可计算的问题，这

是通过证明它等价于通用语言L_{univ}而证明的。波斯特对应问题看上去极像一个叫人绞尽脑汁的算法大赛题目，如果一个编程天才少年没有经过可计算性理论的学习，他很可能在这道题目上徒劳许久。

A.3.4　形式语言与自动机

可以证明：如果存在非递归语言，那么必然存在不能被任何图灵机识别的语言——就是说无法构造一台图灵机，接受属于该语言的所有字符串，且对不属于该语言的字符串拒绝或者不停机。下面证明这个结论。

如果一个语言L和它的补集$\bar{L} = \{\omega \mid \omega \notin L\}$都是图灵机可识别的，那么$L$一定是递归语言。因为若识别$L$的图灵机是$M_L$，识别$\bar{L}$的图灵机是$M_{\bar{L}}$，则可以构造一台新图灵机$M$，它在输入字符串$\omega$上轮流交替地一步一步模拟$M_L$和$M_{\bar{L}}$。因为$\omega$要么属于$L$要么属于$\bar{L}$，所以对于$\omega$，要么$M_L$进入接受状态并停机，要么$M_{\bar{L}}$进入接受状态并停机，于是$M$终将看到$M_L$或$M_{\bar{L}}$之一进入接受状态并停机。这个事件一旦发生，$M$这样做：如果是$M_L$进入接受状态并停机，则$M$进入接受状态并停机；如果是$M_{\bar{L}}$进入接受状态并停机，则$M$进入拒绝状态并停机。可以看出$M$是一个接受$L$的判定器，即$L$属于递归语言。所以如果$L$不属于递归语言，则必然$L$或者$\bar{L}$不可被图灵机识别。

确实存在非递归语言，如刚刚证明的L_{univ}，所以确实存在不可被图灵机识别的语言。L_{univ}是图灵机可识别的，因为通用图灵机 UTM 接受L_{univ}，则L_{univ}的补集——语言$\bar{L}_{\text{univ}}$不是图灵机可识别的。能够被图灵机识别的语言称作递归可枚举语言（recursive enumerable language），刚才证明了非递归可枚举语言的存在，那么能否显式地构造一个非递归可枚举语言呢？

说一个集合是“可列的”（countable），是指存在从自然数集到该集合的一一映射。自然数集本身当然是可列的。偶数集也是可列的，因为存在自然数集与偶数集之间的一一映射：$f(n) = 2n$。整数集也是可列的，因为可以依次列出全部整数：$0, 1, -1, 2, -2, 3, -3, \cdots$，将此数列中每一个整数的位置映射到该整数，就是一个从自然数集到整数集的一一映射。如果一个集合可列，则该集合的基数（cardinality number）是$\aleph_0$（阿列夫零，最小的超限基数）。偶数集是自然数集的真子集，自然数集是整数集的真子集，但它们的基数都是$\aleph_0$。

字符集Σ上的全体字符串集合Σ^*是可列的。因为可以首先列出长度为零的字符串——空字符串，再列出全部长度为 1 的字符串，接着是全部长度为 2 的字符串，如此下去就可以列出Σ^*中的全部字符串。图灵机M既然可以被编码为字符串，那么全体图灵机的集合也是可列的。构造一个无穷行无穷列的二维矩阵D，在D的列上依次标注全体字符串ω_j（$j = 1, 2, \cdots$），在D的行上依次标注全体图灵机M_i（$i = 1, 2, \cdots$），如表 A-2 所示。

表 A-2　图灵机与字符串的接受/不接受关系

\	ω_1	ω_2	ω_3	ω_4	ω_5	…
M_1	0	0	1	1	0	…
M_2	1	1	0	1	0	…
M_3	0	1	0	1	1	…
M_4	1	0	0	1	0	…
M_5	1	1	1	1	0	…
…	…	…	…	…	…	…

D的每一个元素$D_{i,j}$取 1 或 0，表示图灵机M_i是否接受ω_j。例如：$D_{3,4}=1$表示图灵机M_3接受ω_4，而$D_{2,3}=0$表示图灵机M_2不接受ω_3。第i行上全体为 1 的列对应的ω_j就构成图灵机M_i识别的语言L_{M_i}。

D的对角线元素是$D_{k,k}$（$k=1,2,\cdots$）。现在构造这样的语言L_{diag}：若$D_{k,k}=1$，则令$\omega_k\notin L_{\text{diag}}$；若$D_{k,k}=0$，则令$\omega_k\in L_{\text{diag}}$。如此构造的$L_{\text{diag}}$与每一个$L_{M_i}$在$\omega_i$上不一致，即$L_{\text{diag}}$不同于任何图灵机$M_i$识别的语言，也就是说$L_{\text{diag}}$是一个不能被任何图灵机识别的语言，它不是递归可枚举语言。L_{diag}被称为对角线语言。

还有另一种证明存在非递归可枚举语言的方法。在一个无穷的 1/0 串前加一个小数点，就形成一个$[0,1]$区间内的二进制实数。每一个二进制实数对应一个语言：第i位上的 1/0 标记字符串ω_i是否属于该语言。也就是说，可以在全体语言集合与$[0,1]$区间实数集合之间建立一一映射，所以全体语言集合是不可列的，它的基数大于$\aleph_0$。而全体图灵机的集合是可列的，全体语言与全体图灵机之间无法建立一一映射，于是必然存在无法映射到某台图灵机的语言。

在递归语言集合的内部还有层次。若请一位精通正则表达式的程序员写一个能匹配所有回文（palindrome，例如“abracadacarba”）的正则表达式，也许会浪费他很多时间，除非他知道这是不可能的。正则表达式（regular expression）能够接受或拒绝特定字符串，即能够识别一个语言，所以它也是一种计算模型。正则表达式的计算能力等价于有穷状态自动机（finite state automaton）。对于某个正则表达式，可以构造一个有穷状态自动机识别该正则表达式识别的语言，反之，对于某个有穷状态自动机，也可以构造一个正则表达式识别它的语言。可以构造式地证明这个结论：对正则表达式的三种基本操作构造对应的有穷状态自动机组件，反之亦然，本书省略此证明。

能被有穷状态自动机识别的语言属于正则语言（regular language）。利用泵引理（pumping lemma）可以证明回文不属于正则语言，所以不可能写出一个匹配全部回文的正则表达式。本书省略泵引理的证明。

回文属于上下文无关语言（context-free language），上下文无关语言是正则语言的超集。给有穷状态自动机添加一个容量无穷，但是只能先入后出访问的栈（stack），它就成为下推自动机（pushdown automaton）。一个上下文无关语言能被一台下推自动机识别。大部分计算机编程语言都属于上下文无关语言。上下文无关文法（context-free grammer）的计算能力等价于下推自动机，编译器进行语法分析用的就是上下文无关文法。

上下文无关语言是递归语言的子集。各类语言的包含关系是：正则语言⊂上下文无关语言⊂递归语言⊂递归可枚举语言⊂全部语言，如图 A-19 所示。

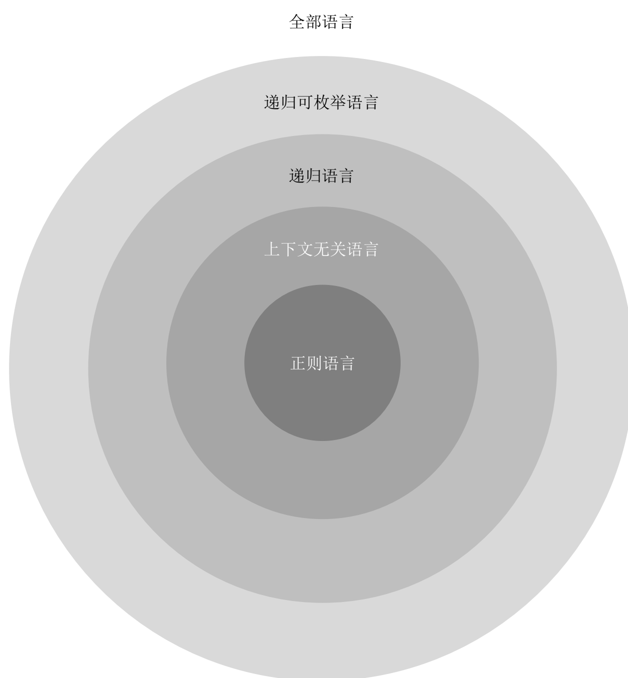


图 A-19　各类语言的层次关系

递归语言代表一类问题，对这类问题存在算法给出明确答案：“是”或“否”。递归可枚举但非递归的语言代表另一类问题，对于这类问题，如果答案是“是”，终会知道；如果答案是“否”，有可能永远无法知道。对角线语言和通用语言这样的非递归语言，代表着无法计算的问题。

A.3.5　停机问题

停机问题（halting problem）的表述是：是否存在判断任意图灵机在任意输入字符串上是否停机的算法？答案是不存在。

假如存在这样的算法，记为Oracle，就可以构造一个图灵机H，当输入字符串为$\langle \widetilde{M}, \omega \rangle$时，$H$先执行Oracle，判断$M$在$\omega$上是否停机，如果不停机，则$H$拒绝$\langle \widetilde{M}, \omega \rangle$；如果停机，则$H$模拟$M$在$\omega$上的执行。如果模拟显示$M$进入接受状态，令$H$接受$\langle \widetilde{M}, \omega \rangle$；如果$M$进入拒绝状态，令$H$拒绝$\langle \widetilde{M}, \omega \rangle$。显然，图灵机$H$是识别$L_{\text{univ}}$的判定器，我们已经知道这是不可能的，所以不存在算法Oracle。

换一种证明方式：如果存在算法可以判断任意图灵机在任意输入字符串上是否停机，那么就可以设计这样一台图灵机，它首先执行这个算法判断自己在输入字符串上是否停机。如果算法的结论是停机，则令该图灵机进入一个死循环永不停机；如果算法的结论是不停机，则令该图灵机立即停机。于是这台图灵机就在它停机的输入上不停机，在它不停机的输入上停机，这产生悖论。

A.3.6　规则 110 的图灵完备性

ECA 也是一种计算模型，将输入数据以某种格式编码为二进制串，以该编码作为初始构型启动 ECA，若 ECA 进入一种构型不再变化则计算完成，最终的稳定构型是计算结果的编码。根据图灵–邱奇论题，ECA 可以被图灵机模拟，但 ECA 是否可以模拟图灵机？即 ECA 是否是图灵完备的？这与具体的 ECA 规则有关。

第一类规则不可能是图灵完备的，它们对任何输入都输出全 0 或全 1 构型。第二类规则也不可能是图灵完备的，在第二类规则下，某一个元胞的值只会受到它的有限邻域内的元胞的初始值的影响，要计算足够大的二进制数的加法，例如$0111 \cdots 111$或者$0111 \cdots 110$加上 1，最右一位是 1 还是 0 会影响最左一位的最终值，在第二类规则下这是不可能的。第三类规则呈现混沌行为，也不大可能是图灵完备的。最有趣的是第四类规则，在第四类规则下，ECA 在瞬态展现自组织行为，能够形成复杂、稳定且相互之间可以交互的结构，这些结构很有希望被用来构造时钟、累加器等组件。

Stephen Wolfram 指出：最有希望具有图灵完备性的是第四类规则，他甚至猜测所有第四类规则都是图灵完备的。Stephen Wolfram 的研究伙伴 Matthew Cook 证明了规则 110 是图灵完备的。本章开头介绍的生命游戏也已被证明是图灵完备的，它属于二维情况下的第四类规则。混沌边缘、自组织、耗散结构和生命等概念与可计算性有着深刻的联系。

A.4　分类、训练与吸引子分岔

元胞自动机的运行使不同初始构型落在不同吸引子上。吸引子把相空间中某个区域内的所有构型都吸引向自己，这个区域称为该吸引子的吸引盆（basin of attraction）。用 CNN 模拟 ECA 的运行，深度越大则运行时间越长，若运行时间长过瞬态，则构型落在吸引子上。对于分类问题，

如果不同类别的样本（其特征向量作为初始构型）随着运行落在不同吸引子上，就可以根据最终的吸引子来判断样本的类别。

若改变 ECA 的规则，则吸引子的数量、形态和性质也会发生变化，那么分类结果就会发生变化。训练 CNN 相当于改变它所模拟的 ECA 的规则。ECA 的规则是离散的、不连续的，无法通过梯度下降来训练，一个办法是用原点斜率较大的 Logistic 函数来近似阶跃函数，但这仍然存在困难。原点斜率较大的 Logistic 函数的饱和区域大，非饱和区域的导数也大，容易发生梯度消失或梯度爆炸。在这里，我们暂且将实际可操作性放在一边，假想存在一种训练 ECA 规则的方法，着重探究规则的改变对吸引子的影响。

对于 ECA 来说，改变规则就是改变动力系统的参数。改变动力系统的参数会导致吸引子的数量和性质发生变化，这称为分岔（bifurcation）。我们看一个动力系统分岔的例子，考虑离散逻辑斯蒂映射（logistic map）：

$$x_{n+1} = rx_n(1 - x_n) \tag{A.6}$$

离散逻辑斯蒂映射在第$(n+1)$时刻的值x_{n+1}取决于第n时刻的值x_n。迭代执行该映射，经过足够的步数（瞬态）后，x_n的值就落在吸引子上，吸引子的性质取决于参数r。图 A-20 展示了当r在[0, 4]区间内变化时吸引子的变化情况。

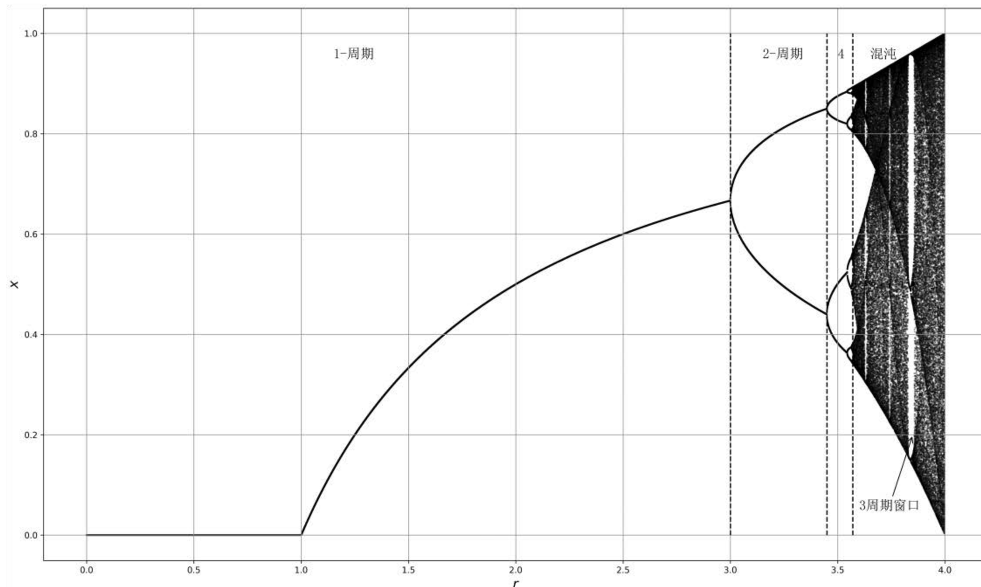


图 A-20　倍周期分岔

图 A-20 是这样生成的：横坐标是参数r的取值，对每一个r值，以 0.5 为初始值迭代执行式（A.6）2300 次，前 1800 次迭代的值不显示，作为瞬态被忽略，将后 500 次迭代的值画在图中。坐标系中每一条竖线上的点是在特定r值下，瞬态之后 500 次迭代的值，它们都在吸引子附近。若竖线看上去只有一个点，其实是 500 个点都落在几乎同一个位置。若竖线看上去有两个点，其实是 500 个点几乎都落在这两个位置上——吸引子是一个包含两个点的循环，即 2-周期极限环。

从图中可以看出，当r处于[0.0, 3.0]区间时，吸引子是一个稳定点，其位置随着r值变化。此阶段吸引子的性质不变，没有分岔。在$r = 3.0$处，吸引子由一个稳定点变成 2-周期极限环。在$r = 3.45$处，吸引子变成 4-周期极限环。随着r增大，吸引子依次成为 8-周期极限环、16-周期极限环……在$r = 3.57$处，吸引子终于成为奇异吸引子，系统表现出混沌行为。

离散逻辑斯蒂映射的吸引子的这种变化称为倍周期分岔（period-doubling bifurcation）。离散逻辑斯蒂映射以倍周期分岔的形式，从秩序走向混沌的过程称为“通向混沌的倍周期通途”（period-doubling cascade to chaos）。有趣的是，系统进入混沌后偶尔还会恢复秩序，出现稳定的周期循环。图中可见，在$r = 3.83$处出现一个 3-周期窗口，将这个 3-周期窗口放大，可以看到系统再次经由倍周期通途走向混沌。将图 A-20 的[3.83, 3.9]区间放大，如图 A-21 所示。

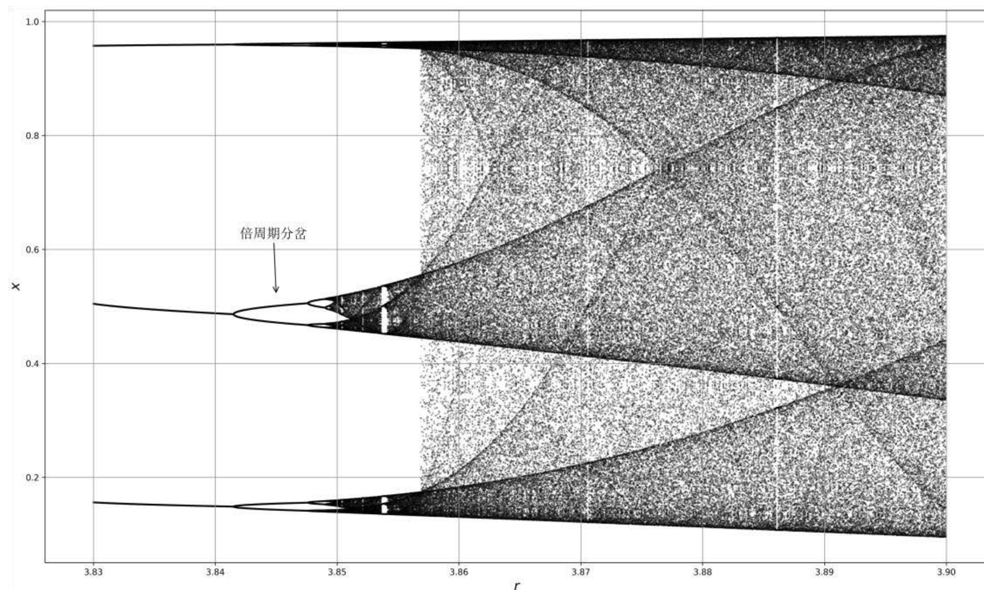


图 A-21　3-周期窗口放大

图 A-21 中可以看到，3-周期极限环的三个点分别发生三个倍周期分岔。注意混沌区域的浅色竖条，那是更小尺度上的稳定周期窗口，它们仍由倍周期通途回到混沌。将图 A-21 中箭头指

向的局部放大，如图 A-22 所示。可看到图中又有很多混沌中的周期窗口，这些周期窗口再发生倍周期分岔回到混沌。偶发的秩序被混沌吞噬，ECA 第三类规则偶然出现的空白区域也是同一个现象。

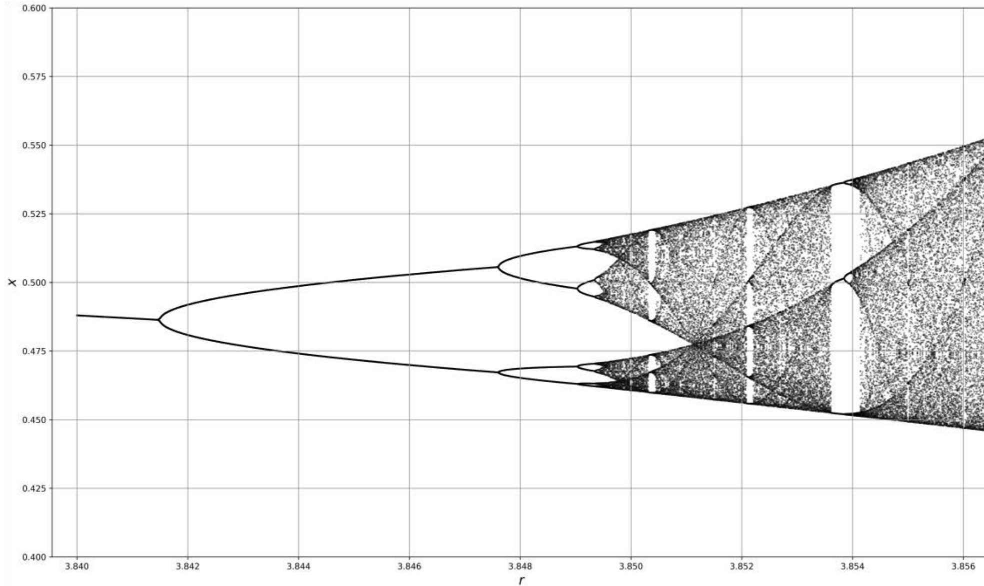


图 A-22　3-周期窗口中部的倍周期分岔

发生倍周期分岔的吸引子始终只有一个，它从稳定点变化为 2, 4, 8, …等偶周期极限环，最终成为混沌吸引子。而分类模型希望有多个吸引子，每个吸引子的吸引盆只包含同类别的样本（初始构型）。我们再考察一种叉状分岔（pitchfork bifurcation），考虑连续动力系统：

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - x^3 = x(\alpha - x^2) \tag{A.7}$$

考察系统的静止点，即导数为零的点。只有静止点才有可能是吸引子，因为系统一定会离开非静止点。当参数$\alpha < 0$时，系统只有一个静止点$x = 0$，此时若$x > 0$，则$\frac{\mathrm{d}x}{\mathrm{d}t} < 0$，点向负方向运动；若$x < 0$，则$\frac{\mathrm{d}x}{\mathrm{d}t} > 0$，点向正方向运动，说明$x = 0$是一个稳定点吸引子。当参数$\alpha > 0$时，系统有三个静止点：$x = 0, \pm\sqrt{\alpha}$。导数$\frac{\mathrm{d}x}{\mathrm{d}t}$的情况是：

- 当$x < -\sqrt{\alpha}$时，$\frac{\mathrm{d}x}{\mathrm{d}t} > 0$，点向正方向运动；
- 当$-\sqrt{\alpha} < x < 0$时，$\frac{\mathrm{d}x}{\mathrm{d}t} < 0$，点向负方向运动；
- 当$0 < x < \sqrt{\alpha}$时，$\frac{\mathrm{d}x}{\mathrm{d}t} > 0$，点向正方向运动；
- 当$x > \sqrt{\alpha}$时，$\frac{\mathrm{d}x}{\mathrm{d}t} < 0$，点向负方向运动。

说明$x=0$是排斥子，$x=\pm\sqrt{\alpha}$是吸引子。吸引子随着参数α的变化如图 A-23 所示。在$\alpha=0$处，静止点$x=0$失去稳定性，吸引子分裂成两个：$x=\pm\sqrt{\alpha}$。分岔图状如叉子，这就是叉状分岔名称的由来。

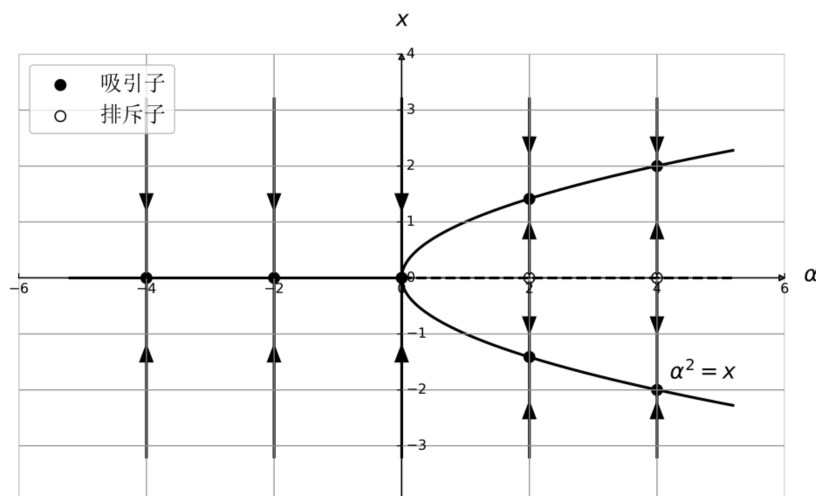


图 A-23　叉状分岔

当系统只有一个吸引子时，所有点（样本）都被吸引向这个吸引子，系统无法区分类别。若随着训练，系统的参数发生变化，吸引子分裂为两个，这时相空间被划分成两个吸引盆，如果两类样本正好位于不同的吸引盆中，则系统就具备了区分两个类别的能力。若两个吸引子还不够，那么使吸引子继续分裂，则相空间被进一步更细地划分。可以想象当吸引子足够多时，相空间被划分成许多不相交的吸引盆，如果每个吸引盆中只包含同类别的样本，则根据吸引子就可以区分类别。

该过程与决策树的生长类似，决策树的叶节点将样本空间划分成不相交的区域，可根据叶节点判断样本的类别。动力系统的吸引子分裂，相当于决策树的叶节点分裂，这两个过程都将样本空间划分成更细的区域，每个区域包含更纯的样本。预测时，样本从决策树的根节点下降到叶节点需要时间，动力系统经过瞬态落入吸引子也需要时间，动力系统的运行时间表现为神经网络的深度，在这层意义上，神经网络的深度与决策树的深度存在联系。

神经网络的训练与吸引子分岔还有一个相似之处，训练神经网络时会观察到一种常见的现象：损失函数平缓地稳定一段时间之后突然下降，然后再稳定一段时间后再次突然下降，训练以间断平衡式的方式前进，这种突变很像是由吸引子分岔导致的。“间断平衡”是古生物学家 Stephen Gould 提出的一种生物演化模式，在物种演化历史中可以观察到这种稳定和突变交替的模式。损

失函数衡量神经网络的适应度，形成选择压力，神经网络的训练与生物进化背后存在着类似的动力学机制，也就不会很奇怪了。

A.5　小结

卷积层与元胞自动机的相似性是显而易见的——它们都根据局部邻域的值确定中心点的值。多个相同卷积层的前向传播可以模拟元胞自动机的运行，我们基于此做出了一些推测。在此，深度有了一个意义：运行时间。深度越大，则其模拟的元胞自动机运行越久。元胞自动机的第四类规则在瞬态具有自组织行为，能够涌现复杂而持久的结构，这些结构之间能够交互，（可能）具备图灵完备性。多卷积层需要足够的深度来模拟足够的运行时长，让元胞自动机有充足的时间完成计算。

卷积层的训练改变卷积核的参数，这相当于调整其所模拟的元胞自动机的规则。训练过程调整规则的不可逆程度，使系统离开接近平衡态的第三类规则，但又不至于离开太远而落入第一、二类规则。训练过程需要将规则调整至第四类，在混沌边缘寻找能够完成分类任务的动力系统。我们可以观察到，训练过程的确会呈现间断平衡的态势。从优化损失函数的角度看，这是梯度下降遇到平缓或陡峭区域，从动力系统的角度看，这体现了吸引子的渐变和分岔。

牛顿爵士说：“我不杜撰假说。”显然我们杜撰的假说已经够多了，最好到此为止。本章权当抛砖引玉，只希望能对读者有所启发。深度学习如此惊人和奇妙，我们不可能不对其背后的深层原因抱有好奇心，毕竟庞加莱说过：“如果世界不是美的，那么它就不值得认识；如果世界不值得认识，那么生命也就不值得经历。”

参 考 文 献

[1] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. 1998, 86(11): 2278-2324.

[2] Krizhevsky A, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012.

[3] Simonyan K, Zisserman A J. Very deep convolutional networks for large-scale image recognition. 2014.

[4] Szegedy C, Liu W, Jia Y Q, et al. Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.

[5] He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[6] Hastie T, Tibshirani R, Friedman J. The elements of statistical learning. Springer series in statistics New York, NY, USA, 2001.

[7] Chong E K, Zak S H. An introduction to optimization.John Wiley & Sons, 2013.

[8] Bertsekas D P. Convex optimization theory. Athena Scientific Belmont, 2009.

[9] Hagan M T, Demuth H B, Beale M H, et al. Neural network design. Pws Pub. Boston, 1996.

[10] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press Cambridge, 2016.

[11] Wolfram S. A new kind of science. Wolfram media Champaign, IL, 2002.

[12] Strogatz S H. Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering. CRC press, 2018.

[13] Hirsch M W, Smale S, Devaney R L. Differential equations, dynamical systems, and an introduction to chaos. Academic press, 2012.

[14] Sipser M. Introduction to the Theory of Computation. Thomson Course Technology Boston, 2006.

[15] Hopcroft J E, Motwani R, Ullman J D. Introduction to automata theory, languages, and computation. 2001, 32(1): 60-65.

[16] Penrose R, Mermin N D. The emperor’s new mind: Concerning computers, minds, and the laws of physics. AAPT, 1990.



微信连接



回复“深度学习”查看相关书单



微博连接

关注@图灵教育 每日分享IT好书



QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

图灵社区
iTuring.cn

在线出版，电子书，《码农》杂志，图灵访谈

机器学习有着艰深的理论背景，不能透彻理解原理，就难以在实践中自由地运用。当广大工程师和学生试图进入机器学习领域时，数学原理是横亘在面前的一座大山。本书就将数学原理、编程实现与实际应用紧密结合起来，为读者引导一条翻越这座大山的通途。

首先，本书以神经网络为线索，沿着从线性模型到深度学习的路线，串起全部核心知识点。我们在必要和恰当的时机引入相关数学基础内容，具有理工科背景的读者能够容易地回忆起相关知识，而不需要再回头求诸于大部头教科书。

其次，除了对原理的讲解，本书也包含编程实现。我们基于Python和Numpy库实现了多种训练算法、二分类/多分类逻辑回归模型以及多层全连接神经网络。另外，本书还实现了一个简单的计算图框架，并展示如何使用该框架搭建和训练多种结构各异的神经网络。

如果你是下面的某一类读者，那么本书就是为你准备的：

- **渴望进入机器学习，特别是神经网络/深度学习领域的高年级本科生和研究生**，本书帮助你将数学基础知识与机器学习和神经网络结合起来，透彻理解其原理和实现；
- **希望了解神经网络与机器学习的广大程序员和工程师**，你可能需要花更多力气回忆数学知识，本书包含了所需要的全部知识点；
- **对机器学习模型的编程实现感兴趣的读者**，本书包含逻辑回归、多层全连接神经网络以及计算图框架的Python实现；
- **对工业界的机器学习、数据科学和数据挖掘工程师**，本书关于模型原理的高级主题，特别是关于模型自由度与偏置-方差权衡方面的内容，可为你提供一些深刻而有趣的洞见。

图灵社区：iTuring.cn
热线：(010)51095183转600
分类建议 计算机 / 人工智能 / 神经网络
人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-51723-4
定价：89.00元